An Explicitly Structured Control Model for Exploring Search Space: Chorale Harmonisation in the Style of J.S. Bach

Somnuk Phon-Amnuaisuk

Doctor of Philosophy
Centre for Intelligent Systems and their Applications
Division of Informatics
University of Edinburgh
2001

Abstract

In this research, we present our computational model which performs four part harmonisation in the style of J.S. Bach. Harmonising Bach chorales is a hard AI problem, comparable to natural language understanding. In our approach, we explore the issue of gaining control in an explicit way for the chorale harmonisation tasks. Generally, the control over the search space may be from both domain dependent and domain independent control knowledge. Our explicit control emphasises domain dependent control knowledge. The control gained from domain dependent control enables us to map a clearer relationship between the control applied and its effects. Two examples of domain dependent control are a plan of tasks to be done and heuristics stating properties of the domain. Examples of domain independent control are notions such as temperature values in an annealing method; mutation rates in Genetic Algorithms; and weights in Artificial Neural Networks.

The appeal of the knowledge based approach lies in the accessibility to the control if required. Our system exploits this concept extensively. Control is explicitly expressed by weaving different atomic definitions (i.e. the rules, tests and measures) together with appropriate control primitives. Each expression constructed is called a control definition, which is hierarchical by nature.

One drawback of the knowledge based approach is that, as the system grows bigger, the exploitation of the new added knowledge grows exponentially. This leads to an intractable search space. To reduce this intractability problem, we partially search the search space at the meta-level. This meta-level architecture reduces the complexity in the search space by exploiting search at the meta-level which has a smaller search space.

The experiment shows that an explicitly structured control offers a greater flexibility in controlling the search space as it allows the control definitions to be manipulated and modified with great flexibility. This is a crucial element in performing partial search over a big search space. As the control is allowed to be examined, the system also potentially supports elaborate explanations of the system activities and reflections at the meta-level.

Acknowledgements

I would like to express my gratitude to:

Dr Alan Smaill and Dr Geraint Wiggins for their guidance and support throughout my research study. The shape of this research owes a lot to the comments and suggestions given by them.

Dr Andrew Tuson for his support during the experiment with Genetic Algorithms in the early period of this project.

Dr John Kitchen, Dr Raymond Monelle and Mr Peter Nelson for giving invaluable comments on the harmonisation output produced by the system.

Dr William Alexander who injected many ideas and concepts during the formative stages of this work and is always happy to look at, and comment on, the harmonisation examples.

Mrs Kimie Okada for her friendship and kindness during my stay in Edinburgh.

Mr Stephen Doughty for many useful discussions on harmony and chorales.

My colleagues in the University of Edinburgh who are always helpful, especially Ben Curry, Luke Phillips and Thomas Segler who helped correct my English in many chapters.

This research has benefited from previous work carried out in the AI and Music group in the University of Edinburgh. I also found the joint M^3 seminars between the Division of Informatics and the Faculty of Music very beneficial. They provided a lot of insights and background to this research.

I would also like to express my gratitude to: my parents, brothers and sisters who constantly supported me during these years; my father-in-law who is willing to read through the very first draft of my thesis; my wife and my son for their understanding of my time away from home and their support which is so valuable in my life.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Somnuk Phon-Amnuaisuk*)

To my parents.

Table of Contents

Chapter 1

Introduction

Research in Artificial Intelligence covers a wide range of interests. Modelling human expertise by a machine is one of the main areas. Systems such as *PRESS* [Bundy and Welham, 1981], *MECHO* [Bundy *et al.*, 1979], *MYCIN* [Buchanan and Shortliffe, 1984], for example, represent the dawn of intelligent machines. Other domains modelled by expert systems include agriculture, business, law, military. The aim of the present research is to model human musical expertise in a chorale harmonisation task. This is not the first time that chorale harmonisation expertise has been modelled in a machine. The *CHORAL* system [Ebcioglu, 1993] and the *HARMONET* systems [Hild *et al.*, 1991] are two examples which fulfil the same role. However, the approach and techniques employed in this dissertation are different from other approaches. We believe our *explicitly structured control model* contributes usefully to the research in this area.

Our research has its roots in an experiment on harmonisation with Genetic Algorithms [Phon-Amnuaisuk, 1997; Phon-Amnuaisuk and Wiggins, 1999]. Subsequently, the need to have more control over the quality of the solution gradually led us to take an alternative path. In this research, we emphasise the idea of controlling the search space at the meta-level where the control structure applied is explicit. In our work, the input problem is a score of the soprano line and the complete solution is the score of the four part harmonisation.

1.1 Modelling Compositional Process

The global view of our research is the study of how to model a human music compositional process using computers. The task involves two fundamental challenges: firstly,

computers must be given access to musical concepts used by musicians; secondly, computers must be equipped with some mechanisms which would allow them to perform some reasoning on these concepts. These two requirements are addressed in three main points:

- The first and second points concern how to capture *musical knowledge* so it can be used by computers. How do we put what we know into a form which allows effective cooperation between human and machine? The knowledge contents should allow effective manipulation. This involves the issues of *knowledge elicitation* and *knowledge representation.*
- The third point concerns how to devise a system which will provide an effective and powerful means for expressing compositional processes. We argue in our research that the arrangement of compositional processes is analogous to the traversing of a search space. This part of our problem may be viewed as how to devise a system which allows a natural means of expressing *domain knowledge* together with *control knowledge* to exploit the domain knowledge.

These three parts of our work are basic knowledge engineering issues which will be discussed in chapter 5 and chapter 6 in this dissertation.

1.1.1 Our disposition towards compositional process

How do we extract knowledge regarding musical composition processes? Although music composition is a subject taught in music education, the creative process behind each musical composition is not usually taught. Sloboda [1985] points out that from the vast literature on musical compositions, the majority deals with musical relationships of the finished products, not the compositional processes. He suggests four plausible methods of acquiring a psychological understanding of compositional process: studying sketches, interviewing composers, asking composers to think aloud and observing composers' improvisatory performance. Our domain knowledge may be seen as woven indirectly from a mixture of methods proposed by Sloboda. We extract knowledge relevant to compositional processes from:

- General practice and idioms in chorale part-writings which have evolved and been established as guide lines.
- Music theory in terms of musical structures and their functions.

The compositional process is focused on activities which we are aware of, think about and act according to, which include both theoretical and empirical dimensions of the domain knowledge. In our model, we devise a number of basic atomic operations dealing with certain aspects of musical knowledge (we call these *atomic control definitions*). The idea is to make these atomic entities capture useful basic ingredients regarding the compositional process; using these, a variety of compositional processes can then be expressed in terms of the same set of atomic constituents.

1.1.2 Forms of control knowledge

To enable machines to act intelligently, the machines must be able to manipulate the domain knowledge they possess. This raises the issue of how to control the exploitation of knowledge. There are two dimensions of control knowledge: domain dependent control knowledge and domain independent control knowledge.

- The domain dependent control knowledge reflects the domain itself. For the chorale harmonisation domain, this would be knowledge of formal structures, harmonic structures of chorales, part writing techniques and other knowledge relevant to chorale harmonisations.
- The domain independent control knowledge includes domain independent control associated with particular search techniques (e.g. cross over rate in Genetic Algorithms) and domain independent control which normally arises from a need to manipulate knowledge represented in a particular implementation.

We believe intelligence may be simulated in machines as the result of interactions between the control and the knowledge content of the system.

1.1.3 Aims of the research

This research aims to investigate the use of explicit control in a knowledge based approach in chorale harmonisation. We list the following sub-goals which are part of the overall goal below.

- To devise a control language that allows flexible exploitation of musical knowledge in a given knowledge base.
- To identify an appropriate knowledge base for chorale harmonisation and to identify an appropriate descriptions of control for this task.

- To show this approach does indeed provide flexible control and gives good quality output.

1.2 Our approach

Since our goal involves the emulation of chorale harmonisation behaviours in computers, it is reasonable to claim that in computing systems, the composition ideas are at a coarser grain size than many lower level operations which, for example carry out simple arithmetic and logic operations. It is the operations at the compositional process level that we want to discuss and control. In our dissertation, structures of the compositional process are discussed at the meta-level. The meta-level control determines the application of the compositional process which is represented with three kinds of basic atomic control definitions (*atomic rules, atomic tests* and *atomic measures*). We express compositional ideas as atomic rules; we control the quality of our solution by the means of atomic tests and atomic measures. A similar approach can be seen in the *PHOEVOS* system [Athanasiou, 1995] which classifies these basic operations as rules, tests and heuristics.

Below, we list the key concepts employed in constructing the system: the *problem reduction principle, modular structure, multiple viewpoints* and *meta-level architecture.*

- The problem reduction principle suggests that the whole problem should be divided into many small parts which may be easier to attack than to attack the whole problem at one time. This is a useful paradigm. However, there are problems inherent in it. When each small part is separated from the big chunk, the hierarchy and the dependency in the original form must be maintained.
- The modularity of knowledge is another important concept in representing knowledge. It allows knowledge to be grouped and organised for effective knowledge management.
- Multiple viewpoints is the concept that reminds us that different representations of a phenomenon are useful. This means that the same phenomena may have different valid interpretations. This enhances the expressiveness of the representation.
- Meta-level architecture is a concept which allows knowledge to be expressed in terms of *knowledge about knowledge.* Knowledge about knowledge may be seen as another layer of domain dependent control theory which is built on top of the

existing domain knowledge. The construction of the meta-level theory depends on the object-level theory.

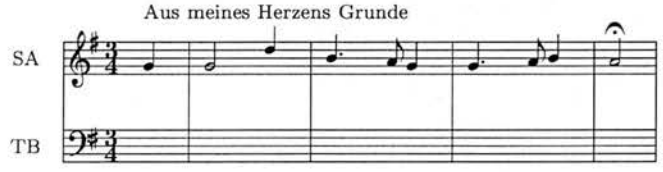
1.2.1 An explicitly structured control model

As the name implies, our *explicitly structured control* model aims to provide control in an explicit way. To achieve this goal, we devise a set of basic control primitives that allow different atomic control definitions to be woven together. The composed control definition reflects a harmonisation process applied to the input score.

1.2.1.1 The Score representation

We devise a data structure to represent musical information and name it a score. The term 'score' used in this dissertation refers to our 'score' data structure. The score has two parts: *music material* and *music interpretations*. The music material part is analogous to the printed score, while the interpretation part is analogous to various interpretations we may have on the music material (see figure 1.1). The harmonisation process (as defined by a control definition) is an operation carried out on this score.

Initially, the music material of the score contains only the input soprano line; and the interpretations of the score contains information such as phrase structure, time signature, key signature, chorale title, etc. Below is the rendition of an input melody (music material) of the given example in *Prolog* notation.




```
line(soprano,[event([5,nat,4],[0,8],[notation:ksig:'G major',
                                     notation:tsig:34,
                                     notation:anacrusis:8]),
              event([5,nat,4],[8,16],[]),
              event([2,nat,5],[24,8],[]),
              event([7,nat,4],[32,12],[]),
              event([6,nat,4],[44,4],[]),
              event([5,nat,4],[48,8],[]),
              event([5,nat,4],[56,12],[]),
```

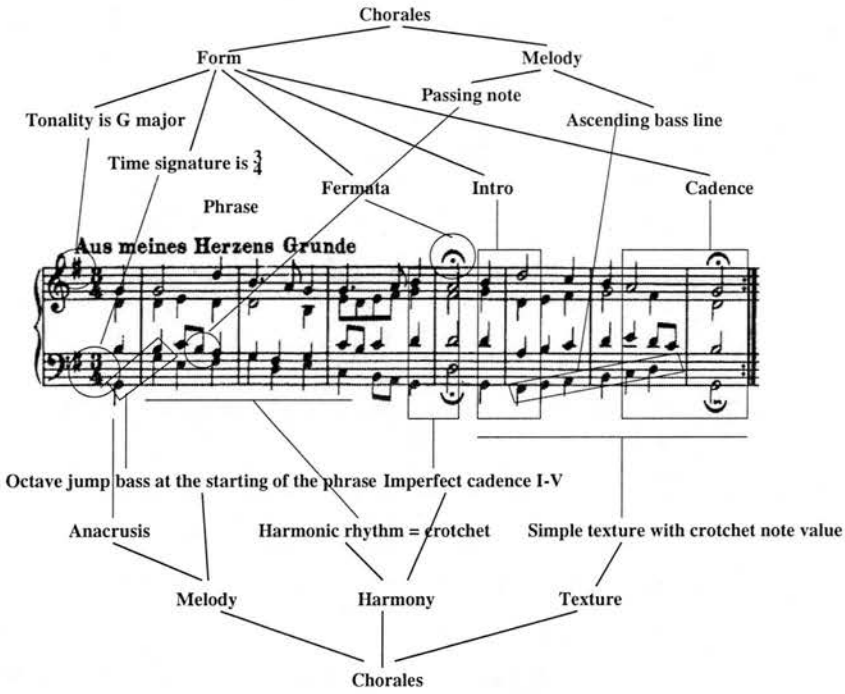


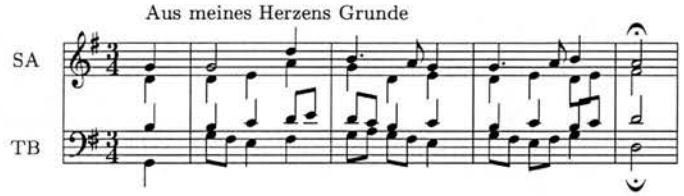
Figure 1.1: Representation of musical information

```
event([6,nat,4],[68,4],[]),
event([7,nat,4],[72,8],[]),
event([6,nat,4],[80,16],[notation:fermata:true])])
```

Snap shots of interpretations at the beginning of the harmonisation process are given below.

- `form:id:chorale` means the object is identified as a chorale object.
- `form:ksig:'G major'` means key signature is in the key of G major.
- `form:tsig:34` means time signature is in $\frac{3}{4}$ time.
- `form:first_strong_beat:anacrusis(8)` means anacrusis duration has the value of a crotchet note.
- `form:phrase_length:[0,96]` means the phrase has an interval span between the time of 0 to 96.
- `harmony:harmonic_rhythm:crotchet` means anacrusis has the duration of a crotchet value.

At the beginning, the interpretations contain minimum information. During the harmonisation process, more interpretations are added and modified until the satisfactory solution is reached. The example below is the harmonisation output of our system, interpretations at this stage contain information of chord progression, outlined voices, etc. Examnples of added interpretations at this stage are given below.



- `form:activePhrase:phrase-1` means the current phrase is the first phrase.
- `melody:nextBassState:state(normal,bass)` means the next bass state is normal (each voice can be in one of these three states: normal, suspension and descending passing note).
- `melody:previousBassState:state(normal,bass)` means the previous bass state is normal.

Interpretations may have complex data structures. In the following three interpretations, the first interpretation shows the detail of the soprano line grouped into different phrases; the second interpretation shows the harmonic outline of the first phrase. Information included here is inversion type, doubling or omitted pitch, chord function in the current key; and the last interpretation keeps the history of harmonic progressions which have been tried on the phrase.

- ```
  melody:phraseGrouping:[phrase(phrase-1,[0,96],
  [[element([notation:ksig:G major],[5,nat,4],[0,8])]:0-8,
  [element([],[5,nat,4],[8,16])]:8-16,
  ...
  [element([notation:fermata:true],[6,nat,4],[80,16])]:80-96])
  ```
- ```
  harmony:outline:phrase(phrase-1,[0,96],
  [sonance(tonic([5,nat,4],[G major,major],
  [[5,nat,0],major,[root_doubling],root_position]),[0,8],
  [s:[pitch([3-3,1],[5,nat,4],[0,8])]:0-8,
  b:[5,nat,2],a:[2,nat,4],t:[7,nat,3]]),
  ```

```
...
sonance(dominant([6,nat,4],[G major,major],
[[2,nat,1],major,[root_doubling],root_position]),[80,16],
[s:[pitch([1-3,1],[6,nat,4],[80,16])]:80-96,
b:[2,nat,3],a:[2,nat,4],t:[4,#,3]])])
```

- `harmony:harmonicPlanHistory:`
  `[[G major-tonic,G major-tonic,G major-subdominant,G major-dominant,`
  `G major-tonic,G major-tonic,G major-subdominant,G major-tonic,`
  `G major-subdominant,G major-tonic,G major-dominant]]`

1.2.1.2 Control language

There are two main components in our control language: atomic control definitions and control primitives. There are three basic types of atomic control definitions: atomic rules (I_R), atomic tests (I_T) and atomic measures (I_M). Control primitives provide the means for connecting various controls definitions together for a more complex control behaviour (e.g. **then**, **and**, **or**, **branch**, **repeat**, etc).

The atomic rules, atomic tests, atomic measures and control primitives provide the means to guide the traversal of the search space. The atomic rules, atomic tests and atomic measures are designed to capture harmonisation processes at a suitable grain size. The lower level control, which does not reflect useful processes, is hidden at the object level. In other words, at this meta-level, the atomic rules reflect thoughts, such as *'outline a chord'*, *'fill in a neighbor note'*, and do not deal with the details in the data structures employed in the system such as *'what is the length of this list?'*, *'increase counter one step'*, *etc.*.

1.3 Our Contributions

This research is rooted in *symbolic AI* and explores the merits of having control explicitly represented. All the key concepts mentioned earlier are the backbone of our system. We now summarise our contributions to research in the computational musical analysis and composition area.

1.3.1 Computational musical analysis and composition

Computer aided musical analysis and composition is a growing research area. Researchers in AI and music have continuously examined the problem of knowledge representation and the modelling of human musical behaviour. Our research offers the following contributions to the research in this area.

1.3.1.1 A natural representation of musical domain knowledge

In our work, we place more emphasis on the natural representation of musical knowledge. We have decided to implement our knowledge representation using a logical language. This facilitates the uses of standard musical terms in our knowledge base which improves readability and makes the program easy to understand. For examples, a soprano line is composed of successions of pitches and silence; a chord is composed of a group of pitches; a perfect cadence is composed of a dominant and a tonic chord. As the knowledge base expands, this approach proves to be very natural for encoding musical knowledge. A detailed discussion of our representation is discussed in chapter 5.

1.3.1.2 A modular program for the harmonisation task

At the meta-level, we promote modularity in the program by means of modular atomic control definitions. This means that components in the earlier definitions may be modified into different control structures with flexibility. Different behaviours can be easily simulated with differently structured control. We also promote the natural representation of musical activities by devising the atomic definitions using understandable musical terms (e.g. fill-in the bass voice, fill-in the passing note).

1.3.1.3 An evaluation from human experts

Our harmonisation examples are given to experts for an evaluation (Our experts are Dr John Kitchen and Dr Raymond Monelle from the music faculty, Edinburgh University). Although there are some points in the harmonisation outputs of which they feel that other alternatives are more musical, they are quite impressed with the harmonisation results in general. They are of the opinion that our outputs are not inferior, but in some respects, they appear to be musically superior to the outputs from other harmonisation

systems (i.e. CHORAL and HARMONET). A detailed discussion of our harmonisation results is discussed in chapter 7.

1.3.2 Basic Properties and Features of our System

We summarise the main properties and features of our system below:

- Modular structure between knowledge entities: The basic units in this paradigm are the compositional process expressed in the form of atomic rules, atomic tests and atomic measures. The rules, tests and measures are implemented at the object level. The exploitation of the object level knowledge is governed by the meta-level control knowledge. Our approach demands that the system is constructed with a clear separation between the object-level and the meta-level control knowledge.
- The connections between the two levels: This is important in order to provide a necessary mapping between the object-level and the meta-level.
- Input/output knowledge encapsulation: The atomic rules, atomic tests and atomic measures encapsulate I/O data in their structures. This data encapsulation minimises users' efforts to keep track of the corresponding I/O data in each atomic control definition. Users simply express the compositional process without worrying about the shape of input and output knowledge.
- Non-volatile musical concepts: Our score representation captures music materials and their interpretations. It is important that musical concepts generated during a process are non-volatile when the process is finished. The concepts should also be accessible by other processes. This may be seen as writing concepts on a blackboard for public use.
- Reduction in complexity of a problem at the meta-level: At the meta-level, the complexity of a problem is reduced to an easy-to-understand metaphor instead of getting bogged down with the details of how the problem is solved. The complexity from lower level control which is essential for manipulating data in the implemented structure is hidden in the object-level.
- Explicitly structured control at the meta-level: The meta-level abstracts the object-level. There is no limit in the number of meta-theories which may apply to an object-level property. This is a powerful and expressive property of the meta-level architecture.

There are other two features which are only partly implemented at the current stage. Only the framework is implemented and extensions could be seen as immediate future work. These two ideas are explained further in chapter 8, 'Further work'.

- Explanation facility: Each control definition is equipped with an explanation facility. This provides information on the current activity and also gives more explanation to How? and Why? questions from users. At the moment, only some atomic definitions have this facility implemented.
- Reflection at the meta-level: This is the most exciting area which has not been explored in this dissertation. We provide a framework for a basic implementation of this idea.

1.4 Outline of the Dissertation

The dissertation is organised into eight chapters and four appendices. A brief description of each of them is given below:

- *Chapter 2:* This chapter gives an overview of the relevant literature. We provide backgrounds on the history of computer music and the elements in knowledge engineering task of various relevant works. We classify two dimensions in our domain knowledge (i.e. musical structures and musical processes) and point out how search control can be improved by explicitly guiding the search. The chapter aims to provide concise background reading for this thesis.
- *Chapter 3:* The motivation for this research is clearly expressed in the case study (i.e. a comparison between the GAs and the rule-based approach). This case study is the experiment carried out in the early stage of this project.
- *Chapter 4:* This chapter discusses the domain knowledge in detail. This involves some musical jargon. Our domain knowledge is extracted from two main areas: the existing empirical and theoretical sources; and the knowledge content in the *CHORAL* system which includes useful heuristics.
- *Chapter 5:* Provides the main ideas about the explicit control paradigm. We discuss knowledge representation of domain knowledge in general before proceeding to the representation of music structure which is the main focus in this chapter. In this representation issue, our focus is on the adequacy and efficiency of the representation framework. We devise the *Score* representation which provides

sufficient knowledge content for representing music structure. The score representation allows different representation forms (interpretations) to be derived from the musical material. As long as (i) the musical materials capture adequate information (which we means all the notational symbols) and (ii) the system has appropriate knowledge to convert the original material to other forms.

- *Chapter 6:* Discusses compositional processes in details. The compositional processes are constructed with explicit control from atomic control definitions (i.e. atomic rules, atomic tests and atomic measures) with appropriate control primitives. We see that theoretical knowledge alone is incomplete for building a knowledge base since parts of the domain knowledge are always missing. In the musical domain, this results from two main factors. Firstly, the generalisation of musical domain knowledge inevitably throws away irregularities from the domain. Secondly, musical processes which are not visible in the finished products are usually neglected. Therefore we try to capture the missing information in our compositional processes. All the atomic control definitions implemented in our system are given in this chapter.
- *Chapter 7:* The chapter evaluates the research in two ways: the harmonisation results produced from the system and the system itself (e.g. architecture, control framework).
- *Chapter 8:* A discussion of advantages and disadvantages of the approach is given in this final chapter. Further developments in various areas which could be extended from the current point are suggested.

We also include four appendices in this thesis. Appendix A lists the control definitions used in producing harmonisation results. Appendix B contains harmonisation examples from our system; we also include Bach's original version for the reader's convenience. Appendix C contains knowledge extracted from the *CHORAL* system. In appendix D, we include Prolog code examples to show the linking between the object-level and the meta-level.

Chapter 2

Literature Review

Introduction

There are many diverse activities in the AI and music fields. The current research, which covers *an explicitly structured control model for searching a solution space* may be described as *applied AI* and *basic AI*, as specified by Bundy [1987]. This chapter gives a review of topics which are relevant to this research. The contents are organised in four main areas:

- A brief history of computer music
- Knowledge elicitation
- Knowledge representation
- Problem solving methods.

Before going into the content, we wish to make clear the terminology used in the discussion.

2.1 Terminology

Knowledge can be viewed from many perspectives. One can talk about knowledge in terms of meta-level knowledge and object-level knowledge, domain knowledge and control knowledge, declarative knowledge and procedural knowledge. To avoid confusion when referring to these terms in our work, we describe the terms as below.

Declarative knowledge and Procedural knowledge: Generally, declarative knowledge is the "knowing that" knowledge and the procedural knowledge is "knowing how" knowledge. To exemplify this distinction, here is an example taken from Way [1994]. The example shows how the knowledge of a circle is represented in two styles: the first in a declarative approach, the second in the procedural approach.

1. A circle is the locus of all points equidistant from a given point.
2. To construct a circle, rotate a compass with one arm fixed until the other arm returns to the starting point.

In thinking about writing a program, the proceduralists would have to have domain knowledge embedded in the procedures in the programs, while the declarativists would prefer to separate domain knowledge from the procedures which are applied to the domain knowledge [Way, 1994; Winograd, 1985].

Domain knowledge and Control knowledge: Generally, domain knowledge is a section of our understanding of the world which is captured in a system. Control knowledge is knowledge that the system uses for exploiting the domain knowledge. Domain knowledge describes what the system knows about the problem. Control knowledge describes how the system makes use of the domain knowledge. Control knowledge can be domain specific or domain independent. The knowledge can be written down in either declarative or procedural form [van Harmelen, 1989b].

Object-level and Meta-level knowledge: Generally, meta-knowledge refers to "knowledge about knowledge" [Bundy, 1990]. The terms meta-level and object-level are used to express relative relationships between levels. For instance, a meta level can be an object level of a higher meta-level. In a knowledge-based system, object-level knowledge conventionally refers to the domain knowledge and meta-level knowledge to the knowledge about the form and structure of the object-level knowledge [Way, 1994]. Davis and Buchanan [1985, p 390] define the concept of meta-level knowledge as:

> "In the most general terms, meta-level knowledge is knowledge about knowledge. Its primary use here is to enable a program to "know what it knows", and to make multiple uses of its knowledge. That is, the program is not only able to use its knowledge directly, but may also be able to examine it, abstract it, reason about it, or direct its application."

Explicitly structured control: The exploitation of the domain knowledge can be implicitly controlled in systems where there is no clear separation between domain knowledge and control knowledge. By explicitly separating control knowledge from the domain knowledge, we can exert a fuller control on the domain with great flexibility. The explicit control is our research interest. In our work, structures in the compositional process are constructed from control definitions together with control primitives in an explicit way. We discuss this issue in detail in chapter 6.

2.2 A Brief History of Computer Music

With the computing technology revolution in the mid 20th century, scientists exploit the power of computing machine for various applications. Computer assisted composition is one of the areas that has attracted much interest.

Iannis Xenakis was one of the earliest composers to experiment with a *stochastic process* in music composition. However, the first stochastic composition implemented on a computer system was by Hiller and Isaacson [1993]. They experimented with a computer generated composition named the *ILLIAC* suite in 1957. The *ILLIAC* suite is one of the earliest attempts in computer aided composition. The work is created by generating candidates randomly, then screening out bad choices according to some preset rules. The composition experiments carried out in the *ILLIAC* project range from *monophonic, four-part first-species counterpoint* to *chromatic structures.*

The *ILLIAC* project is far from an intelligence system in today's sense. Organisations of knowledge in the system and human-machine interactivities are still at a primitive level. However, this project certainly is the first big step in algorithmic compositions and has demonstrated the potential of computer applications in music. This inauguration led to an early programming system called *MUSICOMP* [Hiller, 1970]. *MUSICOMP* consists of libraries of utility subroutines that implement common operations on musical data structure. Cage and Hiller's work titled *HPSCHD* was generated with one of the subroutine in *MUSICOMP* named *ICHING.* A series of experimental works follow this lead: Robert Baker's *CSX-1 study*—an atonal piece composed from systematic permutations of a twelve-tone row; Baker and Hiller's *Computer cantata*—a series of studies.

Early work in computer assisted composition employed mathematical ideas (e.g. probability laws, statistical analysis, stochastic processes, Markov chains, etc.) as a

formalised process for composition. In the late 1960s, Brook, Hopkins, Neumann, and Wright [1993] employed probabilistic methods to produce computer-generated hymn tunes. The process required statistical analysis based on hymn samples, and the result was accomplished by the use of *Markov chains*. In the mid 1970s, John Myhill composed *Scherzo a Tre Voce* see [Hiller, 1970, pp 57-58] using mainly stochastic techniques.

Compositions based on these techniques are natural for modelling as algorithmic processes, but not natural for composers who want stylistic touches in their works. This results from the lack of *flexible control* and expressiveness in the *composition process* generally associated with automated algorithms; as structures in composition process are rigidly pre-defined in algorithms. Improvement in flexibility of control and expressiveness of control are crucial aspect of composition systems.

Apart from the expressiveness of the control, there are also requirements for user friendliness in musical input-output method, for audibility, etc. The development in the early decades blossoms into various focused areas such as musical data input-output purposes (e.g. *DARMS, Common Music, MUSTRAN, SCORE* see 'Beyond *MIDI*' [Selfridge-Field, 1997]); sound synthesis purposes (e.g. Max Mathews' *Music N* family, *GROOVE*, William Buxton's *SSSP*) and automatic composition generating purposes (e.g. *MUSICOMP*, Koenig's *Project I, Project II*, Barry Truax's *POD*). For those who are interested in the backgrounds of these works, a detailed survey of these works can be found in [Loy, 1989]. Pope [1993] also gives a multi-perspective comparison of these works. For our purpose, we simplify the scenario by viewing all systems in one spectrum:

- At one end, there are needs to experiment with new composition techniques (e.g. different tuning systems, a new sonority, a new rhythmic arrangements which could be declared unreadable if notated with a conventional system). These activities require the systems to be tools for composition. Systems in this category should focus on providing the means to organise as well as experiment with musical thoughts and imagined sound in the composers' heads. The systems in this category may not be pregnant with musical theories.
- On the other hand, there are needs for systems to have more musical knowledge and not merely act as tools to aid composition. Systems with embedded knowledge can help in musical analysis tasks or even to perform automatic composition. It is inevitable that systems in this category must be richly encoded

with knowledge to serve their purposes.

Most systems, of course, do not fall at one end of the spectrum but along the spectrum to a certain degree. The development stream in today's applications see the booming of personal computers with a lot of commercial packages specialising in certain tasks: sound synthesis (e.g. *Csound, Music4C, Chant, MAX*); composition (e.g. *Patchwork, OpenMusic, Symbolic composer*)[1]; music notation softwares which combine the ability of music publishing, sound synthesis, sequencing and editing musical data with the ability to playback *MIDI* files (e.g. *Finale, Cakewalk, Cubase, Sibelius*)[2]. The future trend of these systems appear to be moving towards multimedia applications. They are normally aimed at providing facilities found in studios in a virtual form. However, a system pregnantly encoded with musical knowledge will be the ultimate goal for these systems if they seek new capabilities of an intelligent media (i.e. able to evaluate the quality of the materials at hand).

We shall continue our discussion on the systems with embedded musical knowledge. This will direct the discussion into our research perspective—knowledge-based systems. We will organise our topics according to the development steps in constructing a knowledge-based system: *knowledge elicitation, knowledge representation* and *problem solving methods.*

2.3 Knowledge Elicitation

Knowledge elicitation is the first stage of building a knowledge-based system. It is the process of understanding the problem domain and problem solving activities and how to obtain all that knowledge. The discussion on this issue focuses on the methodologies of how to transfer knowledge from sources to knowledge engineers. In this discussion, we view knowledge elicitation in two ways: firstly, it revolves around how the knowledge is perceived and formed (i.e. in our musical domain, this would be the way music is conceptualised: as a balanced structural form, organic growths, grammars, etc.). It is very important to address this in this domain problem, since it reveals the fundamental constituents of the domain knowledge. Secondly, it involves the techniques and

[1]Information on Chant, MAX, OpenMusic and Patchwork can be found in 'www.ircam.fr'; information on Csound in 'mitpress.mit.edu'; information on Symbolic composer in 'www-ks.rus.uni-stuttgart.de/people/schulz/fmusic'.

[2]Information on these softwares can be found in 'www.codamusic.com'; 'www.cakewalk.com'; 'www.steinberg.net'; 'www.sibelius.com'.

methodologies in acquiring the knowledge. The main aim here is to exercise ways to retrieve knowledge from the sources. Techniques such as *Card sortings* [Gammack and Young, 1984], *Interviewings* [Freeman, 1985], etc., may be employed. These techniques suggest tactics such as asking an expert to think aloud—verbal protocols; using concept cards to sort out how experts see relationships between concepts—concept sortings. It is useful to know these tactics since it is quite common for experts in a domain to declare that they do not know why they do such and such. This is generally the case in a domain such as music, where theoretical knowledge is more like a guide than a theorem. However, we will concentrate the rest of our knowledge elicitation discussion on the first issue: the way knowledge is conceptualised.

Palisca [1980] suggests a very interesting point in his article on *'theory'* that music theory is now understood as principally the study of the structure of music. By the structure of music, Palisca means *melody, rhythm, counterpoint, harmony and form* which include considerations of *pitches, intervals, tonal systems, etc.* at a more fundamental level. There are similar conceptualisations from Erpf and Meyer [Bent, 1987]. Erpf classifies musical analysis into 'constructional analysis', 'psychological analysis and 'analysis of expression'. Meyer [1967] puts them into 'formal', 'kinetic-syntactic' and 'referential' views of musical signification. We argue that the understanding of music—knowledge about music—could be classified as understanding in terms of: *Musical structures* (e.g. musical constructions, its effects on listeners, its meanings and *Musical process* (e.g. composing process, performing process). These two dimensions are the cornerstones of musical computation models.

2.3.1 Musical knowledge as musical structures

Musical theory has developed from a long observation and generalisation of the musical domain. The development of music theory from the treatise of counterpoint techniques by Johann Joseph Fux (1660-1741), the harmonic theory of Jean Philippe Rameau (1683-1764) and the composition techniques of Heinrich Christoph Koch (1749-1816) to *Die Lehre von der musikalischen Komposition* by A.B. Marx (1795-1866) focus around structural organisation, harmony, melody and its decorations. The development of theory always reflects philosophical thoughts at that time. The eighteenth and nineteenth century music theory focuses on the form, structure and organic growth. The development in the twentieth century is much faster and more diversified. We see the

developments of music theory correspond to various philosophical ideas. A new branch of psychology, which emphasises perception rather than on motivation [Bent, 1987], emerges in the early 20th century. Along this development stream, a western music theory is gradually formed.

2.3.1.1 Knowledge of structural constructions

By the late 19th century, the theory of music has been firmly established with these basic building blocks: overtone series of a pitch; organisations of pitches into scales, intervals, chords; and organisations of sound mass series into various musical forms (e.g. *binary, ternary, sonata, etc.*). The ways in which we interpret the relationships of these building blocks (i.e. melody, harmony, rhythm) reveal various knowledge aspects of the constructions.

We start by looking at Schenker's *Fundamental structure.* Schenker (1867-1935), in 1906, proposes one of the most influential development in music theory in his book *Harmonielehre.* He generalises German music from the classical period and claims that he discovers a fundamental structure common in them: the projection of the tonic triad in time. With the transformation of the two-part fundamental structure *(Ursatz)* and the *composing out* processes, the surface of the music emerges from this fundamental structure. Music knowledge in Schenker's view is the hierarchy of events from a very abstract background to a detailed foreground [Bent, 1987, p 81]. Riemann (1849-1919), Schoenberg (1874-1951) and Reti (1885-1957) also see music as a hierarchy of events but the events are linearly developed and grown from a basic unit (the organic growth view). Riemann's *phrase-structure* theory sees musical construction from a fundamental unit which represents a process of growth to a decay of the energy unit [Bent, 1987, p 90]. Schoenberg describes the musical construction as beginning with the motif and in which its rhythmic, intervallic, harmonic, and melodic features are exploited during the growth process [Dunsby and Whittall, 1988, pp 74-85],[Schoenberg, 1967]. Reti's *thematic process* also sees music as growing from motive materials into a higher level, thematic patterns. These patterns recur from movement to movement and form the skeleton of the work [Reti, 1951].

It seems there is no one way to look at music. The variety in music allows it to be viewed through various paradigms. The knowledge of structural constructions could also be expressed in other paradigms: Grammars [Baroni *et al.*, 1982, Grammar of

Bach chorales], [Lerdahl and Jackendoff, 1983, GTTM], [Steedman, 1984, Grammar for Jazz Chord Sequences]; Narration of musical processes as observed in *'A companion to Beethoven's Pianoforte Sonatas'*, [Tovey, 1935]; and other syntactic forms (e.g. *Category and Feature analysis, Set theory, Information theory*) [Bent, 1987]; for examples. In our work, we examine Bach chorales and employ traditional music theory as our knowledge source. The discussion of the domain knowledge is therefore focused on the form and the harmonic structure of chorales.

2.3.1.2 Knowledge of structural effects and structural meanings

It is undeniable that music produces some kind of effect in listeners. But whether the effects are from the meanings inside the music which move the listeners or if it is the listeners' ability to realise the meanings and therefore being moved, is a debatable topic. In the prior reviewed works, we do not mention about semantics, that is the meaning of music. This is partly because the works do not address the issue openly. We believe that theorists are aware of the *effects and meanings* parts of the music, but no one really knows how to attack them.

Kurth (1886-1946) explains the effects of music in terms of tensions which result from energy in a chord and a melody. Hindemith (1895-1963) has the same concept. He devises a theory which has its building blocks in traditional harmony (i.e. overtone series, intervals, chords) but modifies and extends them to fit a new paradigm which is aimed to be applied to music at any period. Hindemith [1942] explains music in terms of the degree of tensions which is related to dissonance levels in chords. With this arrangement, composers can arrange the chords in any desired *harmonic crescendo and decrescendo.* Meyer [1956] addresses the aspect of meaning in his works as the relationships inherent in the musical progress. Meyer sees music primarily as patterns. Upon hearing a piece of music, listeners expect what it might imply. The fulfilment or unfulfilment of the expectation arouses emotions in the listeners' minds. Meyer's concept is further extended in Narmour's *Implication-Realisation model* [Narmour, 1990, 1992].

In the chorales domain, Marshall [1980a] points out that there are aesthetic associations in the use of modes, melody types and texts in Luther's chorales. He suggests that Luther associates the European archetype melody in Ionion mode with enthusiastic texts, and normally uses it with a short-note upbeat. Meditative texts are set

with Dorian or Hypodorian mode with a long-note upbeat Gregorian melodies. Tone painting is also not a foreign idea in Bach's time. In our present work, this dimension of knowledge is not in focus, mainly due to time constraint. There is no doubt that this knowledge is important and provides a better picture of the compositional process knowledge.

2.3.2 Musical knowledge as a compositional process

A compositional process carries an implicit part of knowledge which is not transparent from the finished products. For example, listening to Bach chorales, we can hear many events unfolding in time. The music theory gives us the means to talk about the features of events (e.g. each voice part, cadence types). However, understanding how these events are constructed and being able to construct one are two quite different things. Music composition requires knowledge of composition technique which is not explicit in the finished products. Scholars have been studying sketches in order to realise the approximation of the creative process. However, there is still a lot to learn in this area. Sloboda [1985] suggests four methods which could contribute to psychological understanding of compositional process (i) studying sketches, (ii) interviewing composers or examining what composers say about the work, (iii) observing composers at work, ask composers to think aloud and (iv) observing and describing composers' improvisatory performance (in the case that composers are also performers). Our insight in compositional process is in accord with this suggestion as our domain knowledge incorporates both knowledge from music theory (theoretical knowledge), as well as common practice (empirical knowledge).

Generally speaking, a concept in a composition is related to the way the music is understood (e.g. as a basic unit, and the music grows out from that basic unit, as a balanced form and structure, as a narrative of events and extra musical experiences, etc). Understanding the concept of the work reveals powerful heuristics for a musical composition process. In other words, the composition concept could provide a powerful means to guide a search in the search space. It is our interests to utilise this dimension of the knowledge in our search.

The term *compositional process* used here is not the same as the term *procedural knowledge* which is viewed as a form of representation (i.e. procedural knowledge may be an effective form for representing compositional processes, but compositional

processes can be represented in the declarative form as well). The compositional process reveals composers' intentions at a detailed level and this information is crucial in understanding how the complete work is formed. In our approach, the compositional process is used to guide the search and has a closer meaning to *domain dependent control knowledge.*

In conclusion, musical knowledge is the knowledge of musical structures and the knowledge of compositional processes. The process of acquiring the domain knowledge is knowledge elicitation, which considers the following questions: "What are the basic building blocks of musical structures?", "How are the structures held together?", "What are the meanings of the arrangements of certain structural combinations?", "Are the meanings inherent in the structures or associated with outside contexts?" and "How do we exploit these musical structures?"

2.3.3 Knowledge elicitation approaches

We have presented the formation of musical knowledge in terms of knowledge of musical structure and knowledge of musical process (i.e. harmonisation process in our domain). Knowledge elicitation is generally obtained from two main classes of knowledge sources:

- The knowledge sources referring to an existing theoretical knowledge.
- The knowledge sources from experts' empirical knowledge.

Generally, theoretical knowledge is more economical than empirical knowledge. However, it could also be too general and in many cases it fails to capture detailed irregularities in the domain. Experts' empirical knowledge, derived from the observations of a selected repertoire, is better in addressing irregularities, but the knowledge elicitation process is time consuming and the knowledge base is not as compact as in theoretical form. Thus, in practice, the two forms are often required in an implemented domain knowledge.

One example of knowledge elicitation with the empirical approach is the Choral project of Baroni and Jacoboni [1978]. They carry an extensive study on the first two phrases of Bach's chorale melodies (the study is restricted to chorales in major mode in $\frac{4}{4}$ time and where no modulation occurs in the first two phrases). They conclude with over 50 rules regarding chorale melodies from this study and construct a program for generating chorale tunes using these rules. We list some of the rules here:

- Phrase lengths are normally around 4 to 10 crotchet beats.
- A phrase ends on a strong beat.
- The interval between the highest and the lowest notes within the two phrases never exceeds a tenth (i.e. compound third).

The approach taken by Baroni and Jacoboni [1978] exemplifies the process of knowledge elicitation and we believe the same approach has to be taken for a complete knowledge base. This is laborious work and deserves to be a separate project by itself. Since we are more interested in the control issue, we therefore compromise by:

- Regarding the musical structure: we employ general music theory and also reuse some of the knowledge employed in the CHORAL work [Ebcioglu, 1987]. Our domain knowledge will be discussed in detail in chapter 4.
- Regarding the harmonisation process: we follow guidelines and hints which are generally accepted as healthy part writing practices (see section 4.3.2).

The classification of domain knowledge into musical structure and harmonisation process is based on the nature and content of knowledge. For examples, the encoding of guideline such as 'the bass line should be complete first then the inner part' or 'complete the cadence bass then the opening and then join up the fragments of bass' is very natural as harmonisation processes. However, it is more natural to encode guidelines such as 'avoid the same bass from weak to strong beat' or 'avoid having two high points in the same melody line' as musical structure.

Notice that knowledge classified as harmonisation process cannot be observed from the finished product. Therefore, it provides powerful complementation to the domain knowledge derived mainly from theoretical knowledge.

2.4 Knowledge Representations

What is to be represented in music? The representations can be in the forms of: waveforms (e.g. records, tapes, electronic data files); musical notations; musical text; electronic instructions information (e.g. *MIDI*) or other suitable forms (e.g. predicate calculus for computing purposes). The choice of representation is dependent on the point of view from which the problem is being examined (e.g. traditional harmony, Longuet-Higgins' harmony map [Longuet-Higgins, 1962]). The choice of knowledge representation is also application dependent, since it is impossible to have an absolutely

perfect representation that could satisfy all demands arising in modelling the domain. On the representation issue, Christopher Longuet-Higgins [Desain and Honing, 1992, p 8] shares a profound thought on the issue in his personal communication to Honing:

> "My only comment is to remark that the quality of a representation depends on how well it fulfills the purposes for which it is intended, and to underline the need to specify exactly what these purposes are, and how the representation is to be used in achieving them. A blindingly obvious, but by no means trivial, example is the remarkable efficiency of stave notation for the purpose of sight-reading - a form of representation from which we still have a great deal to learn"

Smith [1985] summarises his view of knowledge representation in his *Knowledge Representation Hypothesis.* This allows us to see the crux of the knowledge-based system: the common ingredients of any mechanical intelligent processes. This hypothesis states:

> "Any mechanically embodied intelligent process will be comprised of structural ingredients that *a)* we as external observers naturally take to represent a propositional account of the knowledge that the overall process exhibits, and *b)* independent of such external semantical attribution, play a formal but causal and essential role in engendering the behaviour that manifests that knowledge."

Brachman and Levesque [1985] point out three basic essential components in a (symbolic) knowledge-based system. They are *the representation language, the inference regime,* and *the knowledge of particular domain.* A system must possess an explicit knowledge base in which an inference regime could access facts implicit in the knowledge base and a knowledge representation language must comply with the above requirements. These requirements have probably never been completely fulfilled, due to the lack of an overall adequacy of the representation language. The adequacy of a knowledge representation language is conditioned by the capabilities of a formal language for encoding knowledge of the world. For example, a spoken natural language is more expressive than a hand sign language. The former possesses a much more powerful syntax and semantics, hence it is capable of giving a better mapping between the real world and the symbols. There are two main issues regarding the adequacy of the representation language:

- Adequacy in expressiveness of the representation language: concerns the ability of the language to describe the world. The expressiveness of a representation

language results from the expressiveness of *primitives* and the expressiveness of the ways these primitives can be assembled.

- Adequacy in conceptual efficiency: concerns the reasoning efficiency as a result of the actual shape and structure of the representation and the degree of human-friendliness the representation language offers [Way, 1994].

In the following discussion, we present a discussion of knowledge representation in two areas. Different knowledge representation schemes will be discussed first. Then we shall investigate different representation systems in the literature.

2.4.1 Logic-based knowledge representation schemes

Imagine rain drops on a window screen: how do they move? How can we model this event in our reasoning system? Leibniz (1646-1716) dreams that "an analysis of ideas could be devised, whence in some combinatory way, truths could arise and be estimated as though by number" [Edwards, 1967, Vol. 4, p. 538]. The invention of logical calculi may help us realise Leibniz's dream in some ways. In the logical approach, knowledge is represented as symbolic objects. The interrelationship between objects can be expressed as functions and relations. Examples of knowledge representation in the music domain would be:

$$\text{G major scale} = \{\text{g a b c d e f}\sharp\}$$
$$\text{scale(G major)}$$
$$\text{tonic(G major,g)}$$

However, at the same time, the logic calculi also expose us to one serious problem: the whole world cannot be captured for reasoning purposes with logic calculi because of the huge complexity of the data. Nevertheless, the logic-based schemes offer a natural and powerful means to capture knowledge and reason about it.

The logic-based scheme takes advantages of a precise syntax and clear semantics from first order logic which represents knowledge about the world in the form of a logical *sentence*. A sentence can be constructed from a single atomic-sentence or complex sentences which are formed by constructing many atomic-sentences with logical connectives. An atomic sentence expresses knowledge in the form of *predicate symbols* or *terms*. Two types of quantifiers, *universal* and *existential*, are used to express properties of the whole collection of objects. Logical inference methods such as *Modus ponens*

and *Resolution* always give sound proofs. Theorem provers and logic programming languages use representation schemes of this kind. The disadvantages of the scheme are associated with problems of: intractability; the restrictions to deductive reasoning; monotonicity and crisp logic.

2.4.2 Alternative knowledge representation schemes

To select an appropriate representation scheme is not an easy decision. Levesque and Brachman [1985] discuss the trade-off between expressive adequacy and conceptual efficiency in various representation formalisms (e.g. logic program form, semantic network form, frame description form). For example, they suggest that full first order logic is more expressive but less appealing computationally than a language in semantic net form. However, we could neither say that expressiveness nor conceptual efficiency is the primary issue and therefore could not say which form is a better formalism. Different knowledge representation schemes offer different strong and weak features. We group other main alternatives below for perusal[3]:

2.4.2.1 Production systems

Production systems represent knowledge in pairs of antecedent-consequent production rules. This exemplifies human problem solving to a certain degree. Control in a production system is from the activation of the antecedent part which will need some kinds of conflict resolution mechanism to decide which production rule to be fired next. The disadvantage of this scheme is that the control knowledge is often mingled with the domain knowledge.

2.4.2.2 Frame systems and Semantic networks

The advantages of this scheme include efficiency of inference tactic [Russell and Norvig, 1995], built-in hierarchy; inheritance, and the graphical appearance which makes them easy to understand. The apparent disadvantage of the semantic net is in its limited expressiveness. The frame is effective only when it possesses concise and organised structures which are hard to achieve.

[3]This is in the same line as what has been generally considered by researchers in AI. [Lugar and Stubblefield, 1989; Russell and Norvig, 1995; Tanslay and Hayball, 1993]

2.4.2.3 Other representation schemes

Data-bases, Graphics, Diagrams and Maps are other forms of representation schemes. The knowledge in database form, which could be in *tables, lists*, is widely used in incorporation with other schemes in the knowledge-based systems. Graphics, diagrams and maps can be inferred by the use of geometric operations on their pictorial representations [Jamnik, 1999]. Geometric representation of musical pitch and harmony have been investigated in many works: a key finding algorithm [Longuet-Higgins and Steedman, 1971], chord progression sequences [Holland, 1989] and perceived harmonic relations [Krumhansl, 1990], for examples.

2.4.3 Issues in knowledge representation

In the knowledge elicitation section (see section 2.3, we suggested that musical knowledge is the knowledge of musical structures and compositional processes. To represent the knowledge, we would like the representation systems to be adequate (i.e. in expressiveness and conceptual efficiency) for representing and reasoning about the application domains. Wiggins, Miranda, Smaill, and Harris [1993] surveys and compares different representation systems under two dimensions: *Expressive completeness* and *Structural generality.* In the survey, the expressive completeness[4] refers to the range of raw musical data that can be represented; the structural generality refers to the range of higher level structure that can be represented and manipulated. Systems such as *Charm* [Harris *et al.*, 1991; Smaill *et al.*, 1993b; Wiggins *et al.*, 1989]; *SmOKe* [Pope, 1992] and *Musical structures* [Balaban, 1992] score high in both criteria and are claimed to be suitable for representing musical knowledge. In our application, we represent musical knowledge in the perspectives of musical structures and harmonisation processes. We feel appropriate, at this point, to give a more elaborate discussion of the following terms: *Adequacy in expressiveness* and *Reasoning efficiency*; *Hierarchy* and *Dependency* in knowledge; and *Explicitness* of representation systems; as the terms will be constantly referred to in further discussion.

[4]The term 'expressive completeness' carries a different connotation from the term 'expressiveness' used in this thesis.

2.4.3.1 Expressiveness and Reasoning efficiency

The expressiveness of different representation systems could vary a great deal due to the nature of the representation system itself. For example, verbose description of a musical passage could be made more expressive and more explicit than a score notation of that musical passage. The conventional musical score is limited to a certain syntax.

Representation schemes are constructed from basic building blocks and more complex knowledge could be derived from them. Generally speaking, the expressiveness of the building blocks, their structures and their composed structures determine the expressiveness and the efficiency of the representation systems. In our musical domain, we could build our musical representation system using the logic based scheme, for example. The basic building blocks may be pitches, time and other appropriate propositions. More complex knowledge representing musical structures and musical processes could be built from these building blocks using predicate calculus. In this case, the expressiveness and the conceptual efficiency of the system emerge as a result of the implementation ingredients (i.e. building blocks employed, data structure employed and the logic-based scheme employed). At the end, neither expressiveness nor reasoning efficiency has more merit than the other. It seems to be a matter of preference.

2.4.3.2 Hierarchy and Dependency

Hierarchy is a crucial aspect of representation. Lugar and Stubblefield [1989] argue that human beings use the *hierarchical problem decomposition* tactic in dealing with everyday life activities. Buxton, Reeves, Baecker, and Mezei [1978] presents the use of hierarchical data structure for computer music. West, Howell, and Cross [1991] presents musical knowledge as musical structure and its subdivision. Balaban [1992] argues that hierarchy and time are fundamental, inter-related aspects in music. Smaill, Wiggins, and Harris [1993a] demonstrates the hierarchical representation of musical structures for analysis and composition purposes. It is natural that some parts of knowledge are conceptually more important than others. Hence, representation systems which offer the hierarchy concept are more effective (in capturing the hierarchical knowledge contents) than those which do not offer the same facility.

In our computation model, we solve a problem by breaking it into many subproblems. This is a good approach, since tackling smaller parts is generally easier than

the tackling of the whole problem at once. However, solving problem by dividing it into many small problems may lead us to another problem called the *dependency* problem. In the body of the domain knowledge, the same structure may manifest different meanings in different contexts. The dependency of knowledge is another crucial property in the reasoning process since the meaning of the knowledge at a global level may change when there are changes at local levels.

2.4.3.3 Explicitness of the representation system

In the AI community, the term 'explicit representation' carries more connotations than just 'being explicit' which could be the impression from the term. Stefik [1995] suggests that an explicit representation has the following properties: *Modularised knowledge; Well-understood semantics;* and *Causal connection between the representation system and system behaviours.* The explicitness of representation systems is another crucial factor in designing a representation system. The higher the degree of explicitness in the representation, the greater are the freedom and flexibility allowed for the knowledge contents to be examined and controlled. This is one of the crucial issues of our research.

2.4.4 Analysis of knowledge representation in the literature review

We will now investigate various music representation systems. Most applications are devised with specific aims (e.g. automatic composition systems, composition tools, sound synthesis, musical notation editing). The knowledge representation system is usually tailor-made to fit its problem solving paradigms. In other words, the adequacy and efficiency of the representation system is the prime concern in most music applications. The structure of the control (i.e. hierarchical structure, explicitness, etc.) in the systems is usually of a lower priority since most applications would trade off an elegant structure with efficiency.

In this section, we illustrate different knowledge representation in some of the composition systems. We examine the specific representation in some early composition systems before we examine the systems which aim to provide a general knowledge representation for musical domain. We start with Myhill's *Scherzo a Tre Voce*, a computer generated algorithmic composition. The musical knowledge is represented in mathematical functions which will determine the musical materials and composition plans. We quote the plan used to generate the themes below.

> "Four of eight defined 'themes' were designated as 'wide', four as 'narrow'. A certain function $\phi_1(t)$ determined the probability that a wide rather than a narrow theme might be chosen t seconds after the beginning of the piece. Narrow themes were favoured at the beginning and end, and wide ones in the middle of the composition. The eight 'themes' consisted of eight functions of a real variable. For example, one of the 'themes' had the form, $\alpha sin\beta t$. This meant that t seconds after this 'theme' had begun, the voice which had this 'theme' had a pitch $\alpha sin\beta t$ semitones above an 'axis' chosen for this particular occurrence of the 'theme'. Likewise, $\phi_2(t)$ was read to be 'fast' or 'slow' just as $\phi_1(t)$ was read to be 'wide' and 'narrow'. Finally, $\phi_3(t)$ was read to be 'complicated or 'simple'." [see Hiller, 1970, p 57]

This representation has obvious disadvantages in that (i) it is very hard to incorporate music theoretical knowledge into its representation language and (ii) it offers little accessibility and control of the knowledge, due to the lack of expressiveness of the chosen representation approach. A more flexible representation approach is seen in *MUSICOMP* [Hiller, 1970] which is designed with the aim to separate procedural logic from specific decisions governing style and structural details (i.e. main program and subroutines). This is a logical step towards a better control of domain knowledge. A higher degree of modularity and hierarchy in the knowledge structure is observed in later composition systems such as *Flavors Band* [Fry, 1991], *Cybernetic composer* [Ames and Domino, 1992] and *CHORAL* [Ebcioglu, 1992], for example. The fact that all of these applications express their knowledge in the form of pitch and time makes the knowledge representation more in line with music theory, hence, a better way to incorporate theoretical knowledge. We give a brief summary of the representations in these systems below:

- The Flavors Band bases its primitives on the pitch-time events. The event streams are manipulated by phrase-processors (e.g. transpose, concat, set events). Flavors Band allows a hierarchical structure in its phrase processor networks. A higher-level behaviour emerges from the chunks of knowledge in different phrase processors and their structures. The control is embedded in a written program.
- Cybernetic composer bases its primitives on rhythm, pitch, chordal scheme and thematic scheme. The chordal and thematic scheme capture knowledge at a higher hierarchical level. Cybernetic composer composes four-layer ensembles: solo part, background chords, bass line, and drums. There is very little awareness between parts, and the control of the composition process appears to be implicitly hard-wired within the program.

- The knowledge representation in CHORAL displays a greater expressiveness from previous examples as it introduces the notion of different view points for the same knowledge contents. We could see the hierarchy of compositional process knowledge embedded within the scheduling of these views (e.g. the chord skeleton view, fill-in view, melodic string view, merged melodic string view and the time slice view). The control knowledge of CHORAL is still implicitly represented in the program.

There are attempts to construct a general representation framework for reasoning purposes. A true general representation framework would imply that the representation is independent from the musical activities one wishes to model. This is a very hard task, since different musical activities may require different focused areas on the adequacy, the structuring of knowledge and the organisation of knowledge in the systems. We give a brief survey of some representation frameworks in the literature below:

- CALM: Blevis, Jenkins, and Glasgow [1992] devises a language called CALM—*A Composition Analysis/Generation Language for Music.* CALM has four basic components: *Musical parameter types, Events, Musical transformation* and *Meta-control expressions.* The first two components describe the musical knowledge and the latter two describe the operations on the described knowledge.
- CARLA: Courtot [1992] represents musical objects as *types* in his CARLA system. Types could be primitive types or composed types. He then defines *methods* which are operations applied to musical objects. These musical objects and methods could be linked together to form a more general musical concept.
- Musical structures: Balaban [1992] emphasises hierarchy and time as the basic ingredients for knowledge construction. Musical knowledge is constructed from musical object and time. Balaban devises musical operators to deal with the musical structures such as concatenation operators which allow new structures being created from the existing ones and time operators which will compute temporal properties of musical structures.
- Wiggins, Harris, and Smaill [1989] represents musical structure in terms of *events* and *constituents.* The constituents allow hierarchical structure organisation of events. The proposed 'abstract representation' refers to the abstraction of pitch and time representation which is not related to any specific data type (e.g. pitch could be expressed as frequency or pitch class).

- SmOKe: The *Smallmusic Object Kernel* [Pope, 1992] is an object-oriented approach for representing music. Musical structure is represented as events. They may be music specific property such as pitches or other arbitrary properties. They may also be grouped to list and nested into trees. SmoKe constructs musical concepts such as chords, ostinati, or compositional algorithms via the packages called *event generators* and *modifiers*.

It is appropriate for systems that focus on conventional composition paradigm to rep resent musical knowledge based on pitch and time. It should be obvious that it is not natural to incorporate standard music theory into the domain knowledge of the representation framework in Myhill's *Scherzo a Tre Voce* (see page 2.4.4).

Many attempts to create a general representation framework for musical computation facilities have been done. Unfortunately, none of the frameworks offer a full implementation at the level where they can be reused with flexibility. This is one of the reasons why every implementation has to start implementing its knowledge representation from a very low level, if not from scratch.

2.4.4.1 Towards our representation framework for explicitly structured control

Our representation framework follows fundamental ideas of events and constituents in the CHARM system. The CHARM events are discrete entities of any kind and they can be grouped into higher level constituents. At the fundamental level of our representation framework, pitch, time and other musical properties (e.g. type properties such as scale, tonality; relationship properties between pitches such as interval, chord; functions such as operation on pitch, time, etc) form musical materials, interpretations and operations. These basic primitives can be hierarchically structured into new properties.

In principle, we describe musical properties and manipulate these properties according to our musical thinking (i.e. harmonisation processes applied to the score). As a matter of fact, it would be quite hard not to see that the representation of musical knowledge naturally falls into musical structures and harmonisation processes since the existing knowledge of musical theory (e.g. theory of harmony, musical forms; composition theory) is developed in concepts that complement this perspective.

We argue that by explicitly representing knowledge in the dimensions of musical structure and compositional process, we address sufficient aspects of music to model musical activities. With reference to the representation of music material and inter-

pretations, we can see that, our representation offers flexibility in the transformation between different data structure (different interpretations) for different computational purposes.

If we devise our representation system with all the issues discussed above in mind, we have in our hand, the key to an effective representation system. Unfortunately, it is not that simple after all. We have to juggle trade-offs between different interests at every decision point along a representation formative process. Each representation system seems to carry some problems associated with their own features. For example, let us suppose our knowledge base is constructed with a hierarchical structure. The following issue of dependency between knowledge units must be brought into attention. In other words, we need to control this dependency. This leads us to the discussion on how to structure the control in order to effectively exploit the domain knowledge within the representation paradigm. In the next section, we examine this control issue by looking at it as a search issue.

2.5 Problem Solving Methods

This section presents problem solving methods in AI. We will focus our discussion on the search issue since all general purpose problem solving methods can be implemented using the search technique; and it is our research interest to implement control for an inference step between one problem state to the next problem state. Thinking of a problem solving step as a transition from one state to the next state is a useful metaphor. In an implementation, we can make the transition process either visible or invisible. Some problem solving techniques often hide the transition activities (e.g. *Connectionism*, *Genetic algorithms*). In such cases, it is not possible to explicitly control or explain the transition activities.

2.5.1 Problem solving as search

Problem solving as search may be seen as a single-agent path-finding problem [Russell and Norvig, 1995; Korf, 1992]. The problem domain is formulated as a set of *states*. The state representation represents and describes the problem in an appropriate concept and representation. To start the search, the state is conceptually/structurally modified by *operators* into new concepts or structures. In AI we use the term *'state space'* to

denote the space of all the states addressable from the starting state by the operators. In real life, a state space of a real problem is very big which makes it impossible to search the whole space. We often have to do a partial search on a real problem. The search is normally evaluated based on four criteria [Russell and Norvig, 1995, p 73]: *Completeness*—it will find a solution, if there is one; *Time complexity, Space complexity*—the amount of resource it uses in the operation; and *Optimality*—whether it gives the highest-quality solution. Search is usually classified into two main types of search techniques: *Uninformed search techniques* and *Informed search techniques.*

2.5.1.1 Uninformed search techniques

Uninformed search techniques employ simple strategies in searching through a state space. The most common uninformed searches are *Generate and Test, Depth first, Breadth first* and *Uniform cost* search. In a simple Generate and Test, the system will simply generate a new state and test if it is a goal state. Once the goal state is reached, the search is complete. The search engine possesses minimum knowledge in this operation (i.e. start state; goal state; how to transform the start state to a new state). The uninformed search is also called blind search since it does not know how far it is from the solution state. The search process is improved in the depth first and the breadth first search technique by implementing appropriate control to keep track of the expansion of new states. This information is used to navigate the search tree; either depth first or breadth first. The search engine may use other techniques (e.g. expand the path with the minimum cost—uniform cost; use iterative deepening search). In real life problem, uninformed search techniques usually take too long to find a solution.

2.5.1.2 Informed search techniques

The most common informed searches are *Hill-climbing* and *Best first* search. We add more knowledge to the search engine to make it more informed, and hopefully more effective. The hill-climbing search knows and could determine the quality of a new state. It always expands the search in a direction which is most promising in its view. Unfortunately, although the hill-climbing search can move towards the solution much faster than the uninformed search, it could get stuck on a small hill. Best first search can avoid getting stuck on a small hill as in the hill-climbing due to its ability to backtrack and try a less promising path if the most promising path does not lead to an

optimum solution.

Information which helps to guide search may be expressed as an evaluation function $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of the path from the initial state to node n, and $h(n)$ is the estimated cost of a path from node n to a goal state. If the evaluation function can correctly determine the $g(n)$ and never overestimate the $h(n)$, then we have the A* search which guarantees completeness and optimality.

Unfortunately, in many real life problems, it is quite hard to establish a perfect informed evaluation function since there may be many equally good solutions which make the determining of a perfect evaluation function very hard; or it may require a huge amount of computation resources for a very sophisticated evaluation function. In this situation, we often lower the optimality of the solution for the sake of computational space and time. In the next section, we will discuss the search control which does not rely on a single evaluation function but is controlled and guided with appropriate knowledge along the search path.

2.5.2 Control knowledge

We have discussed various search techniques in the previous section and appreciated how knowledge can improve the search. Now, let us imagine a huge search tree embedded between a transition from one problem state to the next. The improvement is fundamentally achieved by knowing more about the state space. We could have the states and the operators richly constructed (for example, a hill-climbing technique knows how to determine the quality of the states), the operators knowledge can be fine tuned to know about the hierarchy or the dependency of concepts and the constructions of the states instead of knowing just how to determine the quality of states. The fine grain and richness in knowledge allow us to prune unwanted parts of the state space in an early searching stage. The knowledge of hierarchical structure and the dependency in knowledge unit allow us to construct operators in a way that it performs a more effective search (e.g. early pruning of the search space, minimising backtracking operations in the search) provided that the control structure of the system allows us to control the search as desired.

2.5.2.1 Meta level control knowledge

PRESS [Bundy and Welham, 1981; Sterling *et al.*, 1982; Bundy and Sterling, 1988] is a system for solving algebraic problems. *PRESS* utilises meta-level information about algebraic equations to control search. The meta-level understands the information about the expressions (e.g. argument positions in the equation) and the facts about the properties of the functions occurring in the expressions (e.g. a polynomial function). The meta-level knowledge controls the application of appropriate methods to solve a particular equation type. Davis and Buchanan [1985] give an overview and applications of meta-level knowledge in knowledge-based systems. They examine four areas of meta-level knowledge in *TEIRESIAS* which includes the meta-level knowledge of data structures and reasoning strategies. They conclude that the application of meta-level knowledge supports the ease of maintaining the knowledge base; ease of accumulating more knowledge; and ease of controlling reasoning strategies. Frank van Harmelen [1989b] discusses the use of meta-level inference to control the reasoning process in his PhD thesis. He points out the advantages of separate and explicit control knowledge (in his thesis, he focuses on the use of meta-level knowledge as control knowledge). He classifies the meta-level architectures into three main types:

- Object-level inference: The object-level inference would have all activities taking place at the object level. This is the case of a simple program written without a separation of domain knowledge and control knowledge.
- Mixed-level inference: The mixed-level inference would have both reasoning object-level and meta-level inference but meta-level activities and object level activities are still interleaved. This is true in the case of systems such as *TEIRESIAS*.
- Meta-level inference: This is an opposite side of the object-level inference. The behaviour of the object-level is fully specified at the meta-level, so the meta-level can completely simulate the object level inference process (but in the meta-level). A system from this category is *PRESS*.

Van Harmelen concludes that the meta-level inference is a more attractive choice of implementation than other choices. Among his reasons are: firstly, the object-level knowledge and meta-level knowledge are clearly separated; secondly, control can be explicitly stated and reasoned about; and lastly, the more the reasoning process focuses at the meta-level end of the spectrum, the fuller the advantage of object-level—meta-level separation is obtained since a fuller description of object-level is exploited.

2.5.2.2 Some properties of control knowledge

We argue that in order to search better, we need more knowledge; and in order to utilise the knowledge effectively, we need to manage it for ease of maintenance, development and control of its usage. Among many search techniques, we could draw similarities from various aspects of the search control tactics (these aspects should be viewed as a spectrum). The most general observations are:

- Control knowledge is mixed with the domain knowledge; or is separated from the domain knowledge. The mixture of domain and control knowledge results in a rigid knowledge structure. The separation of control and domain results in a modular structure and flexibility which are desired properties in many cases.
- Control knowledge is either domain dependent or domain independent. The control parameters used in search tactics such as *temperature* in *Simulated Annealing*; *crossover* in *Genetic Algorithms* are examples of domain independent control knowledge. Generally, behaviours from the domain independent control are not as clear as domain dependent control; hence, less clarity in the control concepts is observed.

In our approach, we move forward to the end of the spectrum with: a clear separation between domain and control knowledge; and an exploitation of the domain with domain dependent control knowledge. This approach gives us modularity in knowledge which is a must for further development with a meta-level architecture. This issue is developed in chapters 5 and chapter 6.

2.5.3 Analysis of search control in the literature review

It is a formidable task to think about modelling human musical intelligence in machines. We know almost nothing of human mind and how it works or, ironically, even its existence. However researchers in AI succeed in modelling some aspect of intelligent behaviours (e.g. playing chess–*Deep Blue* has already beaten its human opponents). We could view problem solving methods in symbolic AI as search. A solution in a search space is reached with various search techniques. Knowledge in the system and the way it is constructed determine how effective the search is. Below we discuss aspects of search control in various systems.

Returning to our pioneer, the *ILLIAC* suite, Hiller and Isaacson [1993] employ the stochastic technique in exploring the search space. They then employ some constraints

to select the solution. Using the stochastic technique to generate plausible solutions introduces a huge amount of search branches. A conditional probability, such as Markov chains technique, would prune some of the branches out since parts of the knowledge would now reside in the generation of plausible solutions. The Markov chains technique is also employed by Hiller and Brook's works as mentioned earlier. The Markov chain could be seen as a heuristics for guiding search.

Rothgeb [1993], in his PhD dissertation titled *'Harmonising the Unfigured Bass'* (in 1968), formulates the bass-harmonisation procedures set forth by Heinichen and Saint-Lambert in his computer program. Rothgeb finds that the knowledge in the rule sets is not adequate for the task. This would not be too much of a surprise, as musical knowledge represented in this work is very *local* in nature; it deals with the transition between two events. Most theoretical knowledge taken from theory books is in this form. It is often incomplete and most of the time is in a generalised form. The application of this knowledge depends very much on its context. More expert empirical knowledge is needed to improve the performance of the systems. With this line of thought, we see the incorporation of theoretical and empirical knowledge in Rader's work and Schottstaedt's work.

Rader [1993] writes a program to generate a round. He expresses his domain knowledge in the forms of production rules, applicability rules and weight rules. These rules capture both traditional harmony knowledge and his own empirical knowledge. The program generates an n-parts round by determining its harmony and then melody. Another classic example of heuristics search is from the work called *Automatic counterpoint* by Schottstaedt [1989]. Schottstaedt employs best-first heuristics in his automatic counterpoint system. The domain knowledge in his program is based on the knowledge described in Fux's *'Gradus ad Parnussum'*. He also points out that rule set by Fux is incomplete, and that rules should not be treated with equal weight. He assigns different penalty points to his rules. The search engine uses this information as its heuristic to guide the search. The problem of locality of knowledge is also observed as inadequate judgements about the overall melodic shapes. To address this locality problem, the system must possess a higher level of musical knowledge (i.e. look at a larger view).

Upto this point in our discussion, we can see that theoretical knowledge alone is not sufficient and we must incorporate empirical knowledge to help guiding search. The following works below share one common feature in their knowledge. They strongly

incorporate composition techniques in their problem solving tactics.

Ames and Domino [1992] develop the Cybernetic composer for stylistic composition in four genres: ragtime, standard jazz, latin jazz and rock. In this work, knowledge of how the piece in each genre could be constructed is coded in *a chordal scheme* and *a thematic scheme*. The chordal scheme deals with a phrase structure and its contents (e.g. chord progression, harmonic rhythm, melodic rhythm, etc.). The thematic scheme deals with a global form of a composition and the composing strategy (e.g. the material to be employed in each section). The cybernetic composer implicitly represents musical processes in their various schemes. The ability to backtrack from impasses marks another step in the search control in this work.

Smoliar [1993] views musical structures as processes which are controlled by a schema. A schema is an abstraction of a control structure for music in a particular style. He gives an example of a schema for composing a two-part invention. The control schema invokes THEME, SEQ and FIG subroutines (specific to a two part invention). The THEME, SEQ and FIG are processes and they may be modified with further instructions from the control schema. It is possible that control structures in the schema may produce a piece which is musically inappropriate (e.g. dependencies between two or more simultaneous processes are not satisfied). In such a case, Smoliar proposes a program interrupt mechanism which will provide a debugging for the problem.

Fry [1991] constructs his Flavors Band, a computer aided composition language for jazz and popular musical styles. Fry uses the term 'precision of specificity' to describe his intention aiming at building the system. The 'precision of specificity' means composers are allowed to specify their intention in details. Flavors Band specifies music procedurally. The musical work is constructed in Flavors Band by arranging the *phrase processors* into a tree-structured network. The phrase processors are designed to perform certain tasks. They fall into three categories: *note modifiers* (e.g. to generate events, transpose, harmonise, embellish, etc.); *control flow modifiers* (e.g. to concatenate two event streams, filter, repeat, merge, etc.) and *accessors* to access precomputed event arrays. Musical knowledge is captured in the phrase processors. The knowledge of compositional process is procedurally represented in the arrangements of the phrase processors. The interesting part in the Flavors Band is the users' ability to design and arrange the phrase processors.

Cope [1991] began his *EMI* (Experiments In Musical Intelligence) project in the

early 1980s. He focuses on the idea of the stylistic replication of individual composers. He argues that since each composer possesses a unique style, and music in a tonal period seems to portray many similarities with natural language, Cope sees that a new work could be created from these stylistic knowledge together with the idea of an *augmented transition network* from linguistics. Cope extracts the style signatures of composers from their original works and create new works with these signatures. The process of creating a new work is based on an analysis information of original works and an *ATN*. The analysis reveals patterns common in original works (musical signatures analysis) and other statistical of interested events (musical rules analysis). The analysed signatures are placed in suitable places (based on their places in the original work) in a new work, before the details are filled in. The rules (from analysis information) and the *ATN* determine the proper interpolation of new materials to the fixed signatures in the new work. The *ATN* operates hierarchically by building the new work top-down. In Cope's work compositional process is most of the time accomplished manually.

2.5.3.1 Issues in search control

The above discussion displays three important search control issues which are universal to all knowledge based systems. The issues are spelled out below:

1. The modular structure between domain knowledge and control knowledge: Domain and control knowledge have a tendency to mix with each other. For example, the generate and test in the *ILLIAC* may not have as a clear separation of domain and control knowledge as in *EUTERPE* where control is specified as a schema. However, looking at the schema, it is a mixture of control and domain in its structure.
2. The locality of the heuristics which results from the nature of the heuristics or from the nature of the control of the heuristics: This is a common feature in all systems, Rothgeb points this out in his 'harmonising the unfigured bass'.
3. The insufficiency of music theory as the sole knowledge source: This is clearly illustrated in the works of Rothgeb and Schottstaedt.
4. The incorporation of composition techniques, strategies in their domain knowledge: *EMI* and Flavors Band employ a lot of human decisions in their composition process. This is an effective way to incorporate heuristics into the systems.

However, the structure of heuristic application in the systems is not explicitly constructed for automated processes.

These issues are universal in all systems, we can see that all these works address the search control issues mentioned earlier differently. In our opinion, the *explicit* and *implicit* representations of knowledge are very important issues. When the knowledge is explicitly represented, it allows itself to be examined, modified and reasoned about. This is the fundamental principle for implementing an effective control in a system. In our system, we address these issues as follows:

- Regarding the first issue, we argue that the direction we are heading for is a clear modular structure between domain knowledge and control knowledge.
- Regarding the second issue, the knowledge of hierarchical structure and the dependency in the domain knowledge are employed to structure the search in such a way that the global perspective on the solution is satisfied before the local perspective on the solution. These global perspective and local perspective are dependent on how the knowledge is represented in the system.
- Regarding the last two issues, we see the use of composition techniques as a very powerful heuristics source which should provide sufficient knowledge for our modelling purposes. We aim to support an explicit incorporation of this knowledge in our control structure.

2.5.3.2 Search control in other paradigms

So far, our discussion hovers around a *symbolic* AI perspective. There is another AI approach which has gained popularity in the recent years as an alternative approach—*sub-symbolic* AI. The sub-symbolic AI usually refers to the systems using *GAs* Genetic Algorithms [Holland, 1975] technique and *ANNs* Artificial Neural Network technique. In the early period of this project, we explore the problem using GAs to a certain extent. In this section, we shall discuss the search control in GAs.

The GA approach is inspired by the evolutionary process in nature. With this method, solutions in a state space are represented as *chromosomes*. The first step is to create an *initial population*; these chromosome populations then undergo a *selection process* to screen out unfit chromosomes using the criteria of a *fitness function*. The evaluation of the fitness of each chromosome determines the reproductiveness of the chromosome, the fitter chromosomes will be allowed to produce more offspring. In a

simple GA, genetic-inspired operators like *crossover* and *mutation* are applied to these chromosomes to produce offspring for the next generation. The GA search technique benefits from both domain dependent control knowledge which is encoded in the fitness function, and domain independent control (i.e. the population size, the reproduction operators—crossover and mutation).

The search process in the *ANN* is an abstract idea. We may think about the process of finding appropriate connection weights for each neuron as the search activity. In this scenario, we know that the changes in weights affect the output, but it is impossible to establish a mapping between the control applied to the weight and the output. Both GAs and *ANN* fail to offer a clear control of the traversed path because of their very natures.

2.6 Chorale Harmonisation Systems

Modelling a four part writing expertise has been a popular subject for AI researchers working with the musical domain. Many techniques have been tried and we could say that there is not a single best way. Here, three major techniques: Genetic Algorithms (GAs), Artificial Neural Network (*ANNs*), and a knowledge based approach will be focused on. We choose to discuss the *CHORAL* system which represents a rule-based approach; and the *HARMONET* system which is a hybrid between rule-based and *ANNs*. Regarding GAs, we have decided to discuss our own experiment which illustrates a good motivation and background to our current research. So we feel that it is more appropriate to include the discussion regarding the GAs in chapter 3. We also include some harmonisation examples from the *CHORAL* system and the *HARMONET* system in appendix B.

2.6.1 The HARMONET system

The *HARMONET* system is a result of the "Information Structure in Music" project started by Menzel in 1988 and has undergone a steady improvement till 1997. The literature review here is based on the information in [Hild *et al.*, 1991] and the information from the project information page [5].

[5]Web page: http://i11www.ira.uke.de/ musik/Folien/. The harmonisation examples from the *HARMONET* system illustrated in appendix B is obtained from the same page in the period between October-December 2000.

The *HARMONET* system is a hybrid between a neural network and a rule-based approach. A feedforward neural network is trained with a set of Bach chorales using the backpropagation technique. The training material is extracted from Bach chorales. About 40-50 patterns (windows) are extracted from each chorale. The *HARMONET* is trained with about 9711 patterns of Bach's chorale examples. The representation at the neural net level represents the following information:

- Melodic context (i.e. soprano input)
- Function of chords: Function represents chords and their inversions.
- Position in the phrase: Relative position to the start of a phrase.
- Stress information in a bar

The harmonisation process in the *HARMONET* progresses in this fashion:

- A chord and a bass prediction is given by the neural network.
- The rule-based part of the system fills in the inner voices and adds quaver note decorations.

The harmonisation process starts from the left and moves to the right. As there is no backtracking mechanism in *ANNs*, the music representation is encoded with look-ahead information to cope with some global constraints and dependencies (e.g. to guide the harmonisation to a proper idiomatic cadence).

2.6.2 The CHORAL system

The *CHORAL* system[6] harmonises a given input melody in four parts. It is a representative of the systems which have a mixed domain and control. Ebcioglu [1993], in the mid 1980s, attempted to imitate an expert's expertise in harmonising four part chorales in the style of Bach. He devised the *CHORAL* system which is said to have a competence approaching that of a talented student of music who has studied Bach chorales. The system is constructed with more than 350 production rules, constraints and heuristics. However, it is not constructed with the architecture generally regarded as an expert system architecture (i.e. working memory, production rules and inference engine). Instead, the system is implemented with *BSL* (Backtracking Specification Language) with three main components in its program structure:

[6]Original harmonisation examples from the *CHORAL* system are available in [Ebcioglu, 1987].

- Generate section: This consists of condition—action pairs which have the informal meaning of *IF* conditions are true about a partial solution, *THEN* a new element as described by the actions can be added to the partial solution.
- Test section: This contains constraints which could reject certain assignments of the partial solution.
- Recommendations section: This is a heuristic section which asserts what is desirable about the partial solution.

The *CHORAL* system controls search through the search space with an intelligent backtracking system. Intelligent backtracking allows the backtracking process to backtrack directly to the appropriate step, thus avoids backtracking to all steps as in ordinary chronological backtracking system. To summarise, the *CHORAL* system searches with heuristics provided in the generation section and the recommendations section. The control of the search is therefore mainly driven from the domain dependent control knowledge (i.e. how to generate the solution, and the recommendation heuristics). Search efficiency is improved by the use of the intelligent backtracking.

2.6.2.1 Harmonisation mechanism in the CHORAL system

Domain knowledge in the CHORAL system is implemented using the BSL language. The program is implemented in three main sections: the chord skeleton view, the fill in process (which includes four views in the process: fill-in view, melodic string view, merged melodic string view and time slice view) and the Schenkerian view. The harmonisation process for each melody is a cycle of processes from the chord skeleton view, the fill-in process and the Schenkerian analysis. The harmonisation is carried out from left to right.

In this section, we give a brief summary of the mechanism behind the CHORAL system. The harmonisation starts with the chord skeleton view on the left most soprano note. The chord skeleton view works out each chord for each crotchet note. When input melodies are quaver notes the program takes the pitch on the strong quaver as a harmony note, except in the phrase ending where the input quaver melodies can be marked with SUSP (suspension), NORM (normal) and DESC (descending passing note) to indicate the type of the quaver note[7]. In the generation of all plausible chords, the system cleverly scopes down the search space by:

[7]SUSP, NORM, DESC are CHORAL markers of its input melody.

- Normalising the input melody to the key of C major for all chorales in the major key, and A minor for all chorales in the minor key.
- Limiting modulation to only closely related key.
- Constraining chords used in the system to I, ii, ii_7, iii, IV, IV_7, V, V_7, vi, vi_7, $\flat$VII, vii^o in the major key and i, ii, ii^o, $ii^{\varnothing}_7$, III, III^+, iv, iv_7, IV, IV_7, V, V_7, v, VI, vii^o, $vii^{\varnothing}_7$ in the minor key.

As a result the chord skeleton view can generate all the possible chords in the key and all related keys using the following pitch set $\hat{c}$, $\hat{d}$, $\hat{e}$, $\hat{f}$, $\hat{g}$, $\hat{a}$, $\hat{b}$, $\sharp\hat{c}$, $\sharp\hat{d}$, $\sharp\hat{f}$, $\sharp\hat{g}$, $\flat\hat{b}$. The system deals with harmonic progression using rules to generate possible progression locally between two chords. At the end of each phrase, the system deals with cadence using cadence rules and cadence cliches. Modulation in the CHORAL is allowed only when accidentals appear in the soprano line or have been generated in other voices.

The fill-in process is the process after the chord skeleton view. Ebcioglu explains the fill-in process using four different views. The main advantage of this approach is that different views examine the problem from different angles. The fill-in view (under the fill-in process) now looks at each quaver pitch in the chord skeleton in terms of a pair of even-odd quaver notes[8]. The fill-in view determines the quaver notes decoration in each voice (i.e. passing notes, neighboring notes, suspension, etc) as four interacting automata (each automaton for each voice). To generalise the mechanism in the CHORAL system, we can say that the melodic string view and the merged melodic string constrain the solution according to melodic constraint horizontally and the time slice view constrain the solution according to harmonic constraint vertically.

2.6.2.2 Some drawbacks in the CHORAL system

The following observations are made about the CHORAL system:

- It is true that in most programs the names of constants, variables and functions used are quite hard to understand and follow. The CHORAL system also suffers from this problem.
- The CHORAL system tends to use a procedural representation of its musical knowledge. Knowledge relevant to the same musical concept may be scattered in different parts of the program. The knowledge is also described using its own notations. In our opinion, this is harder to understand than declaratively

[8]even, odd quavers are the terms defined in the CHORAL system.

describe the knowledge using conventional musical terms and notations. Consider the following two rules given by the CHORAL:

"... If the previous state is descending passing note (i.e. the previous even slot was second above the source pitch), then the odd slot may sound the source pitch, and the even slot may sound the target pitch, and the normal state may be entered. [Ebcioglu, 1987, p 278]."

"... As an exception to the above rule, the key can also change on the final chord, under the following circumstances: when the root of the ending chord is one of la,mi and the key,degree attributes of the penultimate and the final chords conform to the patterns (key,V), (key+fourth,V); or (key,VII), (key+fourth,V). When the degree VII is used in the latter context, it must be in the fundamental position [Ebcioglu, 1987, p 251]."

- In the CHORAL system, there are elements of the knowledge which are not strictly expressed in a declarative format, and the system behaviour arises from a complex procedural chain. In this situation, when one wants to re-use knowledge content, it is very hard to preserve the fidelity of the knowledge content if the procedural part cannot be successfully re-expressed in a declarative format.
- Due to the fact that the CHORAL system solves harmonisation problems from left to right, global dependency in the chorale must be encoded at a local level (e.g. to guess the nature of a cadence while the program has not yet reached the cadence in its procedure, the program must be equipped with a good guess). We believe this influences the shape of the encoded knowledge in the system to a certain extent since the knowledge is not encoded in its most natural way. This could contribute to the 'hard to read' encoded knowledge.
- The CHORAL system captures a lot of useful knowledge. However, the harmonisation results seem to lack an overall direction which is an important character in a good harmonisation. We believe this results from the way the CHORAL solves the harmonisation problem (from left to right) while the system still lacks knowledge to deal with some global dependency.

The factors above make the knowledge contents in CHORAL hard to understand and follow. This makes an extension of the CHORAL system very difficult. In appendix C, we attempt to re-describe the domain knowledge in the CHORAL system using conventional musical notations.

2.7 Towards the Explicitly Structured Control Model

In brief, any problem solving technique could be viewed as being based on encoding the problem domain into some form of representation formats, then applying search strategies to find the answers. It is possible for knowledge engineers to control the search process with *domain dependent* and/or *domain independent* control heuristics. This is a legacy of the traditional symbolic AI approach. The same extent of facility would not be possible for systems constructed with other approaches. For example, in the GAs, search is controlled via reproduction operators (e.g. mutation, crossover, selection process) which are the abstract *domain independent* control. The training process in the artificial neural networks may also be seen as the search for the right weights with abstract domain independent control. In both cases, it is very hard, if not impossible, to establish a direct mapping between the control applied and its effects on the system knowledge.

In the explicitly structured control model used in the thesis, search is controlled by the three main fundamental elements: Rules, Tests and Measures (see page 111, 132). The rules allow the transitions between states. The Tests and Measures allow the evaluations of the states. We construct our control structure by weaving the Rules, Tests and Measure together. The finer the grain size of these elements, the better the control one can apply to the search.

2.8 Summary

It is our interests to explore the search with domain dependent control since we can directly evaluate the results of its application in a clearer way than the results from the domain independent control. For example, in GAs, a certain value of population size, crossover rate, or mutation rate may yield a certain quality in the solution. It is obvious that it is harder to conceive the relationships between the solutions and the controls applied to them. A domain dependent control has a clearer relationship between control knowledge and its effects.

We have clearly illustrated that the separation between domain knowledge part and the control part—how the domain knowledge is manipulated—is a worthwhile effort. However, the separation only benefits directly in areas associated with modular structure of knowledge (e.g. in knowledge maintenance; ease of adopting to different

control strategies). Regarding the effectiveness of control, it provides the important structure but more work is still needed to be carried out. In this thesis, we propose three fundamental elements in designing the control of the search: Rules, Tests and Measures. We will discuss these points in detail in chapters 5 and 6.

Chapter 3

Motivations

Introduction

This chapter presents our initial experiment: 'Comparisons between GAs and a Rule-Based system'. This case study should provide a useful background and help to clarify our need to control a search, which is fundamental in our research.

3.1 A case study: Comparisons between GAs and a Rule-Based system

We discuss a comparison between Genetic Algorithms and a rule-based technique in the music problem domain based on an experiment carried out in the early stage of our research [Phon-Amnuaisuk and Wiggins, 1999]. The objective of this experiment is to show that two systems with the same amount of static domain knowledge may not deliver the same quality for their solutions since they employ different problem solving techniques (in which different amount of implicit knowledge may reside). For readers would like to know more about our early experiment in evolving harmonisation, please look at [Phon-Amnuaisuk, 1997; Phon-Amnuaisuk *et al.*, 1999].

Genetic algorithms (GAs) are a family of computational models inspired by the *evolutionary process.* The techniques were invented by John Holland in the 1960s. In the GA method, solutions in a state space are represented as *chromosomes.* The first step is to make an *initial population,* which is normally randomly generated. Chromosomes then undergo a *selection process* to screen out poor chromosomes and to determine the reproductive fitness of the survival chromosomes. This selection is justified by evaluating each chromosome with a *fitness function.* In a simple GA, genetically inspired

operators like *crossover* and *mutation* are applied to these surviving chromosomes to produce offspring for the next generation. The following list summarises the process mentioned above.

1. Create an initial population.
2. Evaluate the fitness of each chromosome.
3. Select chromosomes to breed the next generation according to the selection scheme.
4. Apply GA operators to the selected chromosomes to generate new chromosomes for the next generation.
5. The entire old generation may be replaced with a newly created generation (i.e. generational GA). Otherwise, some *elite* chromosomes may be kept while replacing other poor chromosomes.
6. If an acceptable solution is found in the chromosome population then stop, otherwise go back to step 2 and repeat the procedures for a new generation.

The above procedure is a simple procedure for a standard GA. In real applications, adjustments are freely applied to the above procedures to fit most to specific problem requirements. We start our discussion by describing the main components in our GA system.

3.1.1 Chromosomes

We employ a direct representation in our chromosome representation. The chromosome is represented as a matrix structure. It consists of five fixed length strings in one chromosome. These strings represent information on *key, soprano, alto, tenor* and *bass* strings. Each line represents a string of length N. Other information for GA operation may be attached to the chromosome, if appropriate. For the time being, the fitness value is attached to the above matrix. We summarise the knowledge contents in a chromosome below:

- The chromosome consists of Fitness, Key information (e.g. G major) and Voice information (e.g. soprano, alto, tenor, bass).
- Voices contain information about pitches (i.e. name, accidental, octave register and duration).
- Key information is included in the chromosome since this implementation represents pitch in terms of degree of scale. The pitch will be meaningful only if its key context is known.

- Fitness information is attached to the chromosome for the use in the GA reproduction process.

3.1.2 Selection schemes

The selection scheme affects the performance of GAs. A strong selection scheme can speed up the evolution but at the same time might encourage the domination of the sub-optimum solution. A weak selection scheme, on the other hand, can ensure the evolution of the optimum solution but with more computational cost. We employed three different types of selection schemes in the experiment. They are:

- Fitness-proportionate selection: The expected probability with which an individual will be chosen to reproduce is the proportion of its fitness to the total fitness of the whole population. Two common types of selection are implemented:
  - *Roulette-wheel selection*: Each individual is assigned a slice of a roulette-wheel according to its fitness proportion. The selection is made by turning the wheel N times to select N members to fill the population.
  - *Stochastic universal sampling selection*: Instead of turning the wheel N times, the wheel can be modified to have N pointers and just one turn is made.
- *Ranking selection*: This technique disregards absolute fitness information. The expected number of times an individual will be chosen to reproduce is based on the rank rather than the absolute fitness value. This is to reduce the selection pressure on the few highly fit chromosomes.
- *Tournament selection*: Tournament selection can be seen as a noisy form of rank selection. Tournament selection is more efficient since the operation does not require a population sorting task. The following variations were implemented:
  - modified tournament (marriage selection): Pick the first best found within N tries. If all fail, then use the original one.
  - modified tournament (pick 1 from N, pick M from N): Pick the best 1 or M from N tries.

3.1.3 Reproduction operators

The role of operators in a GA is to guide the search towards solutions. The operators should be designed so that every point in the search space is reachable by applying these operators repeatedly. The use of domain knowledge is applicable in this process. The design of operators will be based on the above intuition. In this implementation, crossover and mutation operators [Michalewicz, 1992] are the main driving force of the operation. One-point crossover and directed mutation methods are employed.

- Crossover: one point crossover.
- Mutate by allowing alto, tenor and bass to move up or down semi-tone or tone or stay the same. The process is random.
- Mutate by swapping two randomly picked voices from alto, tenor or bass. This gives the effect of changing the same chord to different open, closed positions or changing inversion types.
- Mutate to a different chord type. This mutation generates a new chord from soprano and scale data. A chord is built with the soprano note taken as a root note, a 3rd note or a 5th note. Doublings can be in root-doubling, 3rd-doubling or 5th-doubling.
- Mutate the beginning of each phrase to start with tonic root position on a down beat.
- Mutate the end of each phrase to end with a chord in root position.

3.1.4 Fitness evaluation functions

The fitness function combines information from different evaluation criteria. We summarise the fitness functions employed in the experiment in figure 3.1. In our experiment, we employ a system of penalty values in our fitness evaluation. The higher the penalty values awarded the lower the fitness of the chromosomes. In this implementation, we measure the fitness value according to four different aspects of:

1. Fitness on vertical evaluation.
2. Fitness on horizontal evaluation.
3. Fitness of the start and ending cadence.
4. Total fitness which is the sum of all the fitnesses mentioned above.

Requirement	Penalty
No hidden unison	10
All voices are within their own ranges	10
Intervals between Soprano-Alto, Alto-Tenor are not more than 1 octave	10
Interval between Tenor-Bass is not more than 12 diatonic degrees	10
No 2nd inversion chord unless it is a cadential $\frac{6}{4}$	10
No doubling of leading note, prefer root and fifth doubling	10
No crossing between voices	10
No tritone leap	10
Alto, Tenor should not leap greater than P4	10
Bass should not leap greater than P5, but can leap one octave	10
Seventh from dominant seventh must resolve step down	10
No parallel unison, fifth, octave between two voices	10
Harmonic progression preference (Progression, Retrogression, Repetition)	10,1[a]
Opening the phrase with root position tonic on strong beat	100
End the phrase with a cadence	100,20[b]

- a) Strong progression (*e.g.* fifth descent) would be free from punishment. Here, we have decided to punish retrogression and repetition by 1 point punishment and others with a 10 point punishment.
- b) Wrong cadence patterns are punished at 100 points. Right patterns but with improper inversion or doubling would be punished at 20 points. The punishment in cadence is higher than others since we want to get the cadence in the right context before other parts.

Figure 3.1: Fitness functions

3.1.5 Settings of GAs' Control parameters

There is no standard parameter setting method for the GAs. In the canonical GA (binary string, generational), the population size is at least 30, and most of the time it is higher [Reeves, 1993]. Small populations may not provide enough diversity in the population, and therefore, they may jeopardise search effectiveness (poor coverage of search space). The crossover rate in a canonical GA is typically set around 60 percent of the total reproduction operation. The mutation rate is set to a much smaller value than the crossover rate. Normally it is set to the value of $\frac{1}{L}$ [Mühlenbein, 1992], where L is the length of a chromosome.

This implementation employs a direct chromosome representation. The choice of a population size, a crossover rate and mutation rates is not based on the above assumptions. Instead, the parameters are arbitrarily set at the start, and later on,

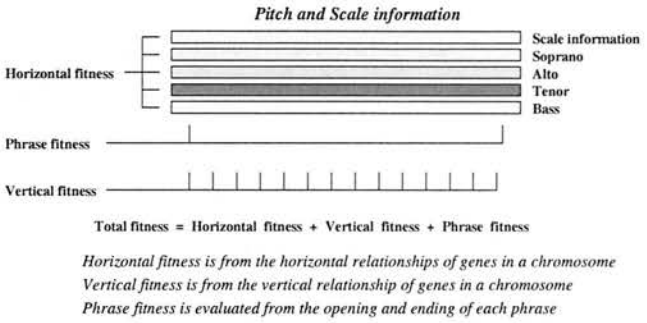


Figure 3.2: Fitness evaluations

they are adjusted to get a suitable setting. Roulette-wheel and Stochastic universal sampling schemes were used in the early development of the program. During the test phrase, Ranking and Tournament selection are used. It is not conclusive (not enough data) from the test data to say which method is a better selection scheme. The main points in the experiment are summarised below:

- The penalty values have a significant effect on the quality of solutions. Other parameters, (i.e. a crossover rate, a mutation rate, different selection schemes), appear to have more effect on the converging time, and do little for the solution quality. This is due to the fact that the penalty values contribute to the knowledge of the system, but other parameters contribute to search paths.
- The uses of direct mutation and an island model speed up the converging time but not the quality of the solution for the same reasons as above.
- The richer the knowledge in the system, the better the quality of the solutions.

3.1.6 Comparison criteria

The experiment is carried out with a chorale style four-part harmonisation. In order to compare the behaviours of two different search techniques, both systems have been constructed with exactly the same knowledge. Both systems employ the same representation structure for musical information. Since both systems have the same granularity in their fundamental knowledge representation level, there is no difference in implicit

knowledge in the knowledge representation at this level. We argue that both systems have the same amount of explicitly coded domain knowledge.

The harmonisation rules employed by both systems are the standard four part harmonisation practices found in literature on harmony. To compare the result of a rule-based search with the GA search, the result of a rule-based search is rated with the same fitness function from the GA system. The fitness function is constructed from the harmonisation knowledge of standard four-part harmony writing procedures. The evaluation is carried out at each vertical, and between each pair of adjacent horizontal, positions. The list below summarises all the requirements and the corresponding penalties if the requirements are not met. The fitness function evaluates the results according to these requirements. The penalties are, in some cases, imposed into multiple levels of punishment:

3.1.7 Results from the case study

We now present the results from both systems with the harmonisation example of the first few bars of *Epiphany hymn* and *Bach Chorale ('Aus meines Herzens Grunde')*. The fitness profile of the best chromosome in each GA generation for 250 generations is plotted. We decided to run the GA for 250 generations since at this stage, the GA is already converged and it is unlikely that it would produce a better answer if we let it run longer. We create a three dimensional plot of penalty values, generation numbers and the position in the chromosome (*i.e.*, this corresponds to position of notes in the melodic string). The plot shows the positions which receive punishment by the evaluation function. It is also clearly shown that the GA tends to fall into a local optimum and it is very hard for the present system to get out of this local optimum. One may argue that the nature of harmonisation problem requires a much more sophisticated and highly knowledge rich GA system to accomplish the task. However, once we have a GA system at that level, the system may not be recognisable as a GA at all. The fundamental issue here, we believe, is that the nature of the problem is not compatible with the GA approach using a direct representation.

Both the GA solution and the rule-based system solution are presented in short scores for comparison. In order to give a clear comparison between a GA search and a rule-based search, the result from the rule-based search too, has also been fed to the GA fitness evaluation function so that both results (from the GA—*broken line* and the

rule-based system—*solid line* in figure 3.4, 3.7) could be compared in terms of penalty value.

3.1.7.1 Epiphany

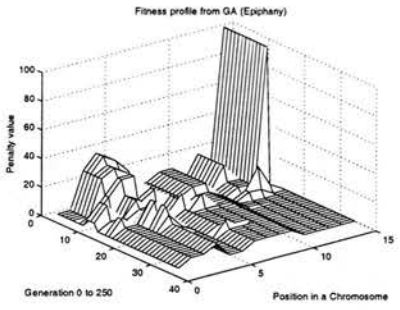


Figure 3.3: GA fitness profile

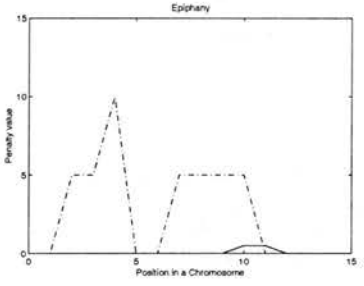


Figure 3.4: Comparison between the GA and the rule-based search

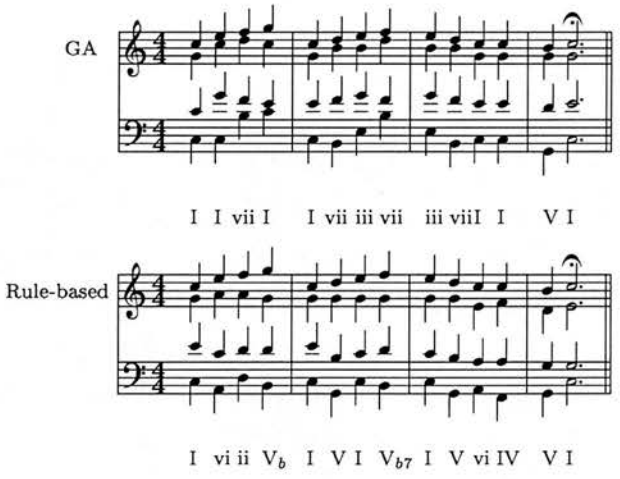


Figure 3.5: Harmonisation results (Epiphany)

The best solution from the GA scores 40 penalty points from:

- Big root bass leap from I to vii progression in the first bar.
- Bass voice on the fourth beat of the first bar is considered as too close to the tenor voice.
- Progression iii to vii is not considered as an appropriate progression.

The solution from the rule-based system has produced a much better output. In this case the penalty is only 1 which comes from soft constraint harmonic progression (*i.e.*, progress from V to vi).

3.1.7.2 Bach's Chorale Aus meines Herzens Grunde

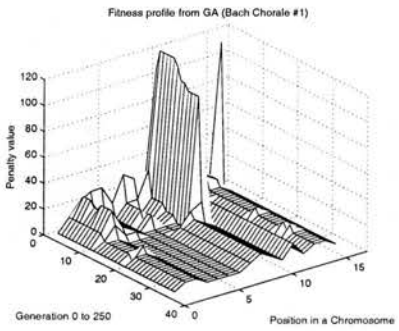


Figure 3.6: GA fitness profile

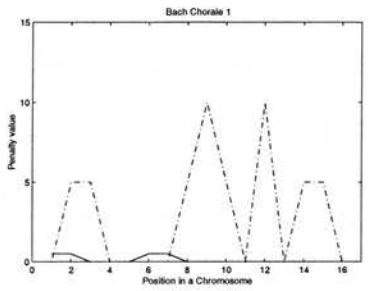


Figure 3.7: Comparison between the GA and the rule-based search

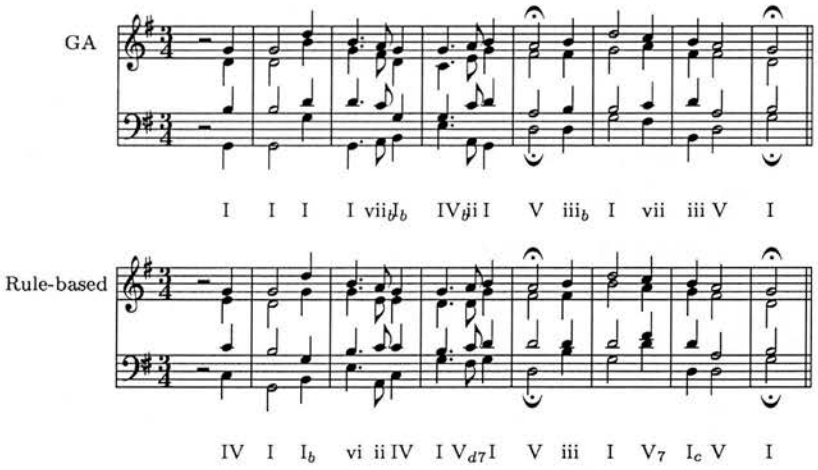


Figure 3.8: Harmonisation results (Aus meines Herzens Grunde)

The best solution from the GA scores 50 penalty points from:

- Big alto leap in the first bar.
- progression ii to I before cadence (first phrase) is considered inappropriate.
- Parallel fifth at first phrase cadence.

- Bass g on the second beat of the second phrase is considered as too close to the tenor voice.
- progression iii to V before cadence(second phrase) is considered as inappropriate.

The solution from the rule based search scores 2 penalty points. It is the penalty from soft constraint harmonic progression (results from progression IV-I).

3.1.8 Conclusion of this case study

It is quite clear from the experiments that the implicit knowledge in the program structure plays a crucial role in contributing to the quality of the harmonisation output. The rule-based system delivers much better output. However, this does not mean that the GA is an inferior system in comparison to the rule-based system. The conclusion from this experiment is that the quality of the output of any system is fundamentally dependent on the overall knowledge the system (explicitly and implicitly) possesses.

This leads us to the idea of promoting the implicit part of the knowledge to be explicit. With explicit knowledge, we should be able to exploit the knowledge more easily. Therefore the system should yield more effective control.

3.2 Summary

We show our motivation in seeking for a way to exert control over a complex search space which could only be partially searched in real life applications. We present the case study which shows that (i) the search structure in the rule-based system carries extra implicit knowledge which is not in the GA system and (ii) the total knowledge (implicit + explicit) determines the quality of solutions.

Chapter 4

Problem Domain

Introduction

In this chapter, we discuss the musical domain problem (i.e. four-part harmonised chorales) in detail. The discussion aims to show how the domain knowledge is developed and formed. We base our discussion on the following topics: a brief history of chorales, the character of Bach's harmonised chorales and the domain knowledge implemented in our system. These topics will inevitably expose the discussion to much musical jargon. We also present relevant domain knowledge extracted from Ebcioglu's CHORAL work (the fill-in view) which is partly used knowledge in our implementation[1]. These topics will be presented in the following order:

- A brief history of chorales
- Bach's 371 harmonised chorales
- Domain knowledge in the system.

4.1 A Brief History of Chorales

Chorales[2] have their root in church music. Marshall [1980a] gives a concise description of chorale in his article in the new Grove dictionary as "The congregational hymn of the German Protestant church service". Hymns and plainsongs are among the types of songs sung in honour of God by Christians. Before the Reformation movement, the

[1]The knowledge base in Ebcioglu's CHORAL work is also included in appendix C.

[2]'Chorale' is an English word with the same meaning as the word 'Choral' which is a German word.

music sung in the Roman Catholic church had become more and more elaborate and could be too complex for ordinary laymen who, if they wanted to take part, had to sing in Latin words and sometimes, in a polyphonic style. The focus on the content of worship and ritual rose to a high peak in this period, and this elaborate polyphonic singing drove away the participation of ordinary laymen. In the 16th century, a new idea which placed the Gospel before the content and the form of worship emerged and came to be known as the Reformation movement. The movement was led by Martin Luther (1483-1546) who also emphasized on the participation of ordinary laymen in church services. He encouraged the congregational singing by introducing *congregational hymns* which were sung in German. The congregational hymn had a simple rhymed metrical verse, memorable melody and was easy to sing. The simplicity of language and music allowed everybody to perform the congregational singing service without special musical or language skills.

Luther's congregational hymn came to be known as *Choral* in the late 16th century. The change may be due to the fact that the congregational hymns were sung in unison in Luther's time. In German, the word 'Choral' comes from Latin words 'Cantus Choralis' which refer to the style of plainsongs in unison [Marshall, 1980a; Baker and Welsby, 1993]. From here onwards, we refer to Luther's congregational hymns as chorales. Luther contributed a great deal to the development of chorales. He supervised and set up texts and melodies of the chorales with materials from various sources. The texts were adapted from Psalms, passages from the Bible and Gregorian chant, for example. The melodies were adapted from Gregorian chant, secular songs, etc. Luther and his musical collaborators—Rhau, Rupsch, Dietrich and Walter, also composed new texts and new melodies [Leaver and Bond, 1980]. The Reformation wave swept through other Christian communities and created more demands for texts and melodies of congregational hymns. The first major chorale anthologies (the *Geystliches Gesangk Buchleyn*) appeared in Wittenberg in 1524 [Marshall, 1980a]. Many other collections with more tunes follow during the active period of the chorale era. In 1738, Johann Balthasar König published *Harmonischer Lieder Schatz* which contained 1,913 tunes.

There is no doubt that the development of the chorales owes a lot to the Lutheran Reformation movement. The austere Reformation approaches such as Calvinism and Pietism jeopardised the development of chorales to some degree. The austere ap-

proaches had little sympathy for art music which they regarded as unwholesome to Christian souls. On the other hand, Luther, being a keen musician, saw music as a gift from God. Thus, his encouragement of congregational singing had no conflict with the use of choirs in church services and the elaborate settings of chorales for voices or instruments.

The active period of the chorale genre could be marked from the start of Luther's Reformation movement in 1517 to the start of the Enlightenment period in 1750. The genre became less and less active towards the end of the 16th century due to war, plague and famine [Marshall, 1980a]. The Thirty-year War in the early 17th century destroyed churches and schools. This encouraged private devotions which were very agreeable with Pietism and the coming Enlightenment thoughts. Only a few melodies were added to the chorale repertoire in the 17th century. The cantata had also become the main form of church service from the beginning of the 18th century in the Lutheran church [Baker and Welsby, 1993]. When Enlightenment thought finally blossomed, the chorale found itself unsuitable to the new thinking. Older chorale texts were either rewritten or removed. The genre finally ended its active role.

Chorales have been a major source of material and inspiration for musicians for generations. Various works based on the chorales have formed a vast repertoire of music (e.g. chorale concerto, chorale motet, chorale cantata, chorale fantasia). In the next section, we focus on Bach's harmonised chorales which are chorales set for four voices.

4.2 Bach's 371 Harmonised Chorales

In Luther's time, vocal settings for chorales appeared in monophonic, polyphonic and homophonic settings. The homophonic settings became more and more popular with the rise of the *cantional style* in the late 16th century. The term cantional style normally refers to chordal four part setting of hymns with melody on the top part. The shifting of the chorale tune which is normally in the tenor part (a general practice at that time) to the soprano part in the cantional style gives the melody line a better clarity [Marshall, 1980b]. This has become general practice up until the present day. We have mentioned that only a few new melodies were added to the chorales during the Baroque period (1600-1750). The contribution to chorales in this period was not from new melodies, but from the four-part harmonisation of chorales. Bach (1685-1750) and his 371 harmonised chorales are regarded as the main monument of the period. The

chorale genre possesses typical characteristics suitable for worship purposes (e.g. a meditative tone). These characters are reflected in the music.

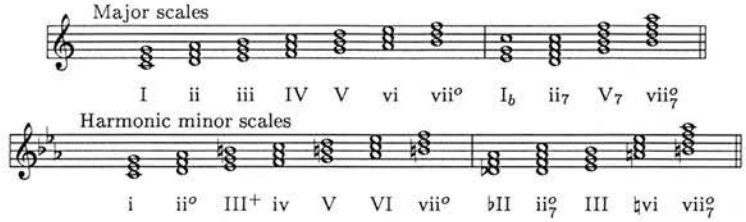
In this section, we introduce the musical knowledge of Bach's chorales. We discuss the knowledge under the following musical concepts: *Form, Texture, Rhythm, Voices* and *Harmony.* Knowledge in this section is extracted from various sources: historical aspects are mostly from [Sadie, 1980]; music theory is from standard harmony theory books [Boyd, 1967; Butterworth, 1994; Gauldin, 1997; Piston, 1982; Schoenberg, 1990; Taylor, 1989; Turek, 1988; Wittlich and Martin, 1989].

4.2.1 Musical symbols used in this thesis

To avoid confusion when referring to musical symbols in our discussion, we provide descriptions of the symbols used throughout the thesis below. The notation system follows the ASA standard (Acoustic Society of America) [Wittlich and Martin, 1989].

- Pitch symbols: We represent to the middle C (on a piano) as C4. Hence, C5 denotes a pitch an octave higher and C3 denotes a pitch an octave lower. Accidentals are put in front of numbers (e.g. C♯4, D♭4).
- Scale degrees: Each pitch in a major scale is represented as a number with a hat on top: $\hat{1}$, $\hat{2}$, $\hat{3}$, $\hat{4}$, $\hat{5}$, $\hat{6}$, $\hat{7}$. The representation of a minor scale is relative to its major scale. Hence, the harmonic minor scale has the following scale degrees: $\hat{1}$, $\hat{2}$, $\flat\hat{3}$, $\hat{4}$, $\hat{5}$, $\flat\hat{6}$, $\hat{7}$. Letters from *a* to *g* are also used to represent the scale degrees, for example, C major: $\hat{c}$, $\hat{d}$, $\hat{e}$, $\hat{f}$, $\hat{g}$, $\hat{a}$, $\hat{b}$; C minor: $\hat{c}$, $\hat{d}$, $\hat{e}\flat$, $\hat{f}$, $\hat{g}$, $\hat{a}\flat$, $\hat{b}$.
- Interval symbols: The following symbols are used: perfect (P), major (M), minor (m), augmented ($^+$) and diminished (^{o}). For example, intervals between $\hat{c}$ and $\hat{e}$ is M3; between $\hat{c}$ and $\hat{f}\sharp$ is 4^+; and between $\hat{c}$ and $\hat{g}\flat$ is 5^o.
- Chord symbols: We use Roman numerals I, II, III, IV, V, VI, VII to refer to chords. The upper case symbols denote major chords, the lower case symbols denote minor chords. Accidentals placed in front of chord symbols denote the accidentals on the root of those chords. Augmented and diminished qualities are indicated by adding ($^+$) to a major chord symbol and (^{o}) to a minor chord symbol. Chord inversions are indicated by subscripts a, b, c, d. I_a denotes a chord I in its root position, I_b in its first inversion, I_c in its second inversion and I_{7d} in its third inversion. Numbers used with chord symbols indicate extra intervals added to the chord root. For example, in a key of C major, I_7 indicates a C major triad

with an added major seventh interval and $I_{\flat 7}$ indicates a C major triad with an added minor seventh interval. We give example of various chord symbols below:



- Other chord notations: There are other symbols used to notate altered chords and to show added notes. Secondary dominants are notated as V/V, vii°/V; neapolitan sixth $\flat II_{\flat}$; augmented sixth 6^{+}It, 6^{+}Gr, 6^{+}Fr (for Italian, German and French respectively). Added notes are used in the same way as in the seventh chord, so V_4–V_3 implies a suspension from $\hat{c}$–$\hat{b}$ in the key of C major.

4.2.2 Form, Texture and Rhythm

Bach composed only a few new chorale melodies. The majority of his 371 harmonised chorales are the reharmonisation of existing chorale tunes used in the Lutheran church. Riemenschneider [1941] points out that Bach's chorales are not furnished for the congregation but are to be sung with instrumental support. The tunes and texts of these chorales range from two line strophes (phrases)[3]. (e.g. chorale R130—*Meine Seele erhebet den Hern*)[4] to over eighteen line strophes in long works (e.g. chorale R132—*Kyrie, Gott Vater in Ewigkeit*; R205—*Herr Gott, dich loben wir*). However, the majority is organised in the range of four to eight line strophes. Each strophe is marked with a fermata and normally ranges from four to ten crotchet beats.

4.2.2.1 Barform

Most chorales are in *barform*[5]. Barform is a three part form composed of a repeated two-line *Stollen* and an *Abgesang* (i.e. AAB form). The Abgesang may be a through composed section, or may conclude with the material from the end of the Stollen. In figure 4.1, we present a digest of the forms observed in the 371 collection.

[3]In the strophic form, different texts are set to the same melody.

[4]The reference numbers used with 'R' in this thesis refer to the chorales based on the Riemenschneider collection.

[5]Brunner [1980] suggests that the word 'Bar' is probably a shortened form of 'Barat', a word taken from the language of fencing and denoting a skillful thrust.

Chorales name (with Riemenschneider ref.	Nos.)	Form
Aus meines Herzens Grunde	1	ABAB|CBAB
Christus, Der ist mein leben	6	ABAB
Ermuntre dich, mein schwacher Geist	9	ABAB|CABB
Ein' feste Burg ist unser Gott	20	ABAB|CDEFG
O Ewigkeit, du Donnerwort	26	AABAAB|BB
Freu dich sehr, O meine seele	29	ABAB|BBCD
Christum wir sollen loben schon	56	ABCD
Wie schon leuchtet der morgenstern	86	ABCABC|DDEF
Helft mir Gott's Gute preisen	88	ABAB|CDDB'
Hast du denn, Jesu, dein Angesicht	90	AA|BCD
Allein Gott in der Höh' sei Ehr'	125	ABAB|CDB'
Keinen hat Gott verlassen	129	ABAB|CBCB
Meine Seele erhebet den Hern	130	AB
Wachet auf, ruft uns die Stimme	179	ABCABC|DDEFC'
Ach Gott, vom Himmel sieh' darein	262	ABAB|CDE
Mach's mit mir, Gott, nach deiner Gut	310	ABAB|CD

Figure 4.1: Examples of form observed in the Bach's 371 harmonised chorales

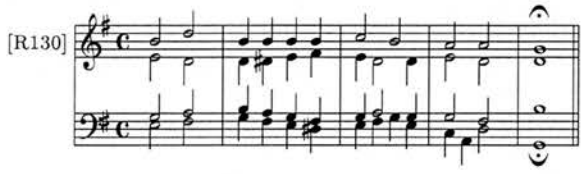
We can see that many tunes cannot be fitted to the barform. For example, chorales R56, R130 show a through-composed character. Chorales generally fall into these two categories: barform, and through-composed.

4.2.2.2 Texture

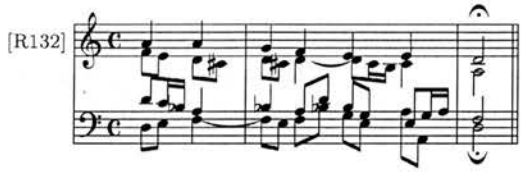
Most of Bach's chorales are set in a homophonic style. The progression comes to a pulse at the end of each strophe (phrase) which is marked with a cadence and notated with a fermata. However, in the performance, some phrases are joined together; and sometimes not all voices come to the fermata at the same time (see chorales R132, R137, R141, R193, R198). Each voice in the chorales normally stays in its own range and rarely crosses another. However, crossings between the alto and the tenor are observed in some chorales (see Bach's chorale R3, R4, R143, R273). Crossing between the tenor and the bass is very rare (see chorale R259). Voicings in the four parts can be in an *open* structure (the distance between the soprano and the tenor is an octave or more) or a *close* structure (the distance is less than an octave). In a four part setting, the

melody and the bass are the most interesting parts of all the parts. A bigger gap is normally allowed for voicing between tenor and bass than those between soprano and alto; or alto and tenor. Boyd [1967] points out that the texture in Bach's chorales may be grouped into a spectrum as follows.

- On one extreme, the texture is a plain, notes-against-notes setting with few or no unessential notes (see chorales R102, R130, R179, R217).



- On the other extreme, the texture shows active semiquaver movements in the parts (see chorales R132, R197, R241).



The majority of chorales fall in the middle of the spectrum with mostly minim, crotchet and quaver movements. Chorale tunes are normally set in a syllabic style with each syllable corresponding to a crotchet. Other parts must be set with sufficient number of notes to be sung with the tunes. A syllable may be held for a longer note value (see chorales R316) but never shortened to a semiquaver which is the shortest note value used in Bach's chorales.

4.2.2.3 Rhythm

The original Lutheran chorales often have a mixture of duple and triple time, and sometimes even in free rhythm. In Bach's time, melody lines have already lost their metrical freedom. The metrical structure and barline have already become the standard musical notation practice. In the 371 collection, the majority is in $\frac{4}{4}$ time while $\frac{3}{4}$ time is not uncommon. Other time signatures are rare. Only one $\frac{3}{2}$ time (chorale R194), and one compound time $\frac{12}{8}$ (chorale R344) are observed. It is not a surprise that the rhythmic structure of the chorale is in a simple form since the exotic rhythmic

pattern has never had a place in Western church music (especially in congregational services). The rhythmic figures of melodies and harmonic rhythms of chorales are expressed in simplicity. Dotted rhythms and syncopation are not common figures for chorales. Harmonic rhythm is always in simple crotchet beat movement.

4.2.3 Voices

In a four-part homophonic setting, parts are assigned for *soprano, alto, tenor* and *bass* voices. The melody is in the soprano part. Other parts help to complete the required harmony. The melodies and texts of chorales move with steady minims and crotchets with occasional quaver passing notes. In this section, we discuss the character of voices and the voice leading techniques in the four-part writing.

4.2.3.1 Chorale tunes

Tusler [1968] points out that the majority of melodies start on a weak beat (*anacrusis*). Only 109 melodies in the 371 collection start on a strong beat (*thesis*). The ending of each phrase is normally on the strong beat with only few exceptional cases. Sometimes Bach weakens the finality by prolonging the fermata over to the weaker beat with the same harmony on the strong beat (see chorales R8, R48, R233).

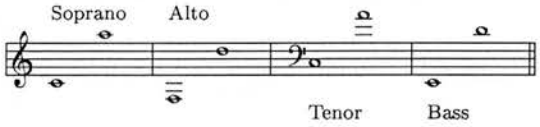
Chorale tunes are distinctive for their simplicity. The melody moves in diatonic steps most of the time, and occasionally with consonant leaps: minor third, major third, perfect fourth and perfect fifth. Leaps of sevenths and tritones are not observed in Bach's harmonised chorale melodies. Leaps of major and minor sixth intervals are also rare in chorale melodies. They appear in the connection between phrases in some works (chorales R20, R21, R49, R68, R210); see [Tusler, 1968, p 20].

4.2.3.2 Voice leading

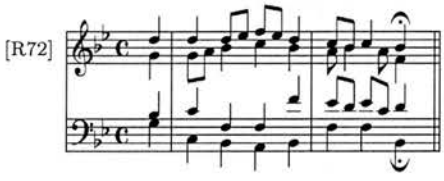
Generally speaking, voice leading concepts in chorales arise from two main factors. The first factor is that chorales are for human voices, not for instruments. Therefore, the voice leading techniques tend to be less adventurous than those of instrumental writing. The other factor is from justifications given to intervals formed by voices (e.g. sweet, firm, hollow, harsh, or even diabolic). The justifications shape their applications. In four part writing, each individual part should be easy to sing. Interesting dimensions could be introduced with varieties in rhythmic figures, texture and harmony. Normally,

the inner parts are more constrained in their movements than the bass part which can be much more active. We discuss general voice leading concepts for chorale part writings below:

- Voice range: The range of each voice is displayed below. Generally, each part should be in its own range with the gap of less than one octave between voices, except for the bass and the tenor voices which could be up to octave and a fifth.

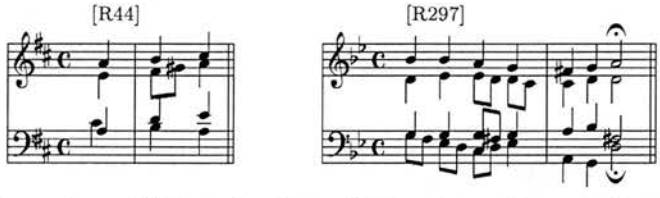


  In Bach's chorales, we sometimes see the bass line going as deep as the ĉ two octaves below middle ĉ (see chorale R70, R131). The gap between alto and tenor is sometimes wider than an octave (see chorale R72 below).



- Consecutives: As a general practice in Bach's day, consecutive fifths and octaves were to be avoided. However, in the 371 collection, there are few examples of consecutives (see chorales R8, R70, R121, R134, R139, R164, R237, R243, R269, R273, R334, R368) [Boyd, 1967].

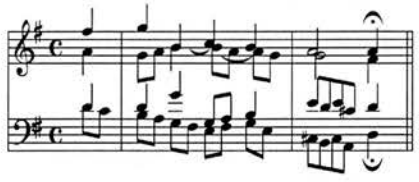
  The progression from perfect fifth to diminished fifth is acceptable, but the progression from diminished fifth to perfect fifth is still taken as a consecutive fifth. Bach seems to be very cautious about consecutives. The examples below are from chorale R44 and R297. It is clear that if there were no rearrangement of the voices, consecutives would occur in both examples.



- Hidden unison, fifth and octave: When two voices reach unison, fifth or octave intervals from different interval types in parallel progression, the progression is named hidden unison, fifth or octave respectively. Hidden fifths and octaves

between the soprano and the bass are not allowed unless the soprano part moves in a step movement. There are no restrictions in these progressions in other voice combinations. However, the use of hidden unison collapses two voices together and there is hardly any gain from this so it is not ordinarily used.

- Movement of inner parts: As a rule of thumb, the alto and tenor always move with the smallest possible interval to the next chord. Leaps greater than a fifth are very rare. However, the example from chorale R9 shows an octave leap in the tenor part.
- Movement of bass line: The bass line in Bach's chorales is more active than the inner parts, leap of third, fourth, fifth are common. The scale movements of the bass flowing in quaver movement with accented or unaccented passing notes is one distinct feature of Bach's chorales. This gives a strong sense of direction to music. An octave bass leap at an anacrusis opening is also a common feature in Bach's chorales.

[R9] 

- Crossing of parts: The alto part never crosses above the soprano. In the 371 collection, the alto and tenor parts are sometimes crossed (see chorale R50, R56, R273) while the crossings between the tenor and bass are rare (see chorale R259). Part crossing is not a common practice and usually happens for a good reason (e.g. avoiding consecutives, obtaining a better line movement).
- Doublings and Omission: As a rule of thumb, doubling is allowed only in: root or 5th in root position major, minor chords; root or 5th in the first inversion major chords; root or 5th or 3rd in the first inversion minor chords; 3rd or 5th in the first inversion diminished chords; 5th only in the second inversion major, minor chords. Never double leading notes. If a note in a chord is omitted, omit the fifth and add extra root or 3rd notes.
- Decorations: The use of non-harmonic tones in suspensions, passing notes, neighbor notes, changing notes, anticipations and retardations helps to give colour to the texture. The 4-3 and 7-6 suspensions are frequently used in the 371 collection while the 9-8 suspension is less frequent. Passing notes and changing notes are

common in all parts. Anticipations are common at the cadence. However, some decoration patterns are uncommon in Bach's chorales. For example, Boyd [1967] observes that the appogiatura is foreign to Bach's chorale style.

4.2.4 Harmony

Harmony normally refers to three or four sounds sounding simultaneously. The sonority from the sound mass could be described as consonance or dissonance. In the realm of tonal harmony, overtone series is the fundamental basis for describing music properties such as intervals and chords. The beginning of '*modern*' tonal harmony is normally regarded as starting from the early 17th century [Dahlhaus, 1980]. About this time, the idea of composition technique with vertical harmony and its progression formed into a new concept. The concepts of chords, chord inversion and relationship between chords were gradually formed. The consolidation of these concepts is now known as tonal harmony.

The majority of Bach's chorales are in major and minor modes. Modulation is mostly confined to the dominant or its relative major or minor. Modal character in some melodies, which are rooted from plainchant origin, is not uncommon. However, Boyd [1967] and Tusler [1968] make an observation that when the melodies appear in modal characters, Bach will not try to imitate the harmony in the earlier style. Instead, he treats them in such a way as to bring them into the tonal system. Bach frequently alters accidentals in a modal chorale to bring it into a minor mode.

4.2.4.1 Harmonic vocabulary

The harmonic vocabulary in Bach's chorales is usually of a simple *functional harmony*. The chords are triads and seventh chords built from major and minor scales. There are three main functional classes of chords:

- Tonic function (T): I, i
- Subdominant function (S): ii, ii^o, iv, IV, V, vi, VI, V/V, vii^o/V, ♭II, 6^+It, 6^+Gr, 6^+Fr, etc.
- Dominant function (D): V, vii^o

First inversion chords are commonly used. The second inversion chords are restricted in the context of passing chords, prolongation chords and cadential $\frac{6}{4}$ chords. The common

seventh chords are those formed from supertonic, subdominant, dominant and leading chords.

4.2.4.2 Harmonic progression and modulation

Harmonic progression in tonal music is governed by *functional tonality*. The principle of this system is that music revolves around focal tonal centres which are established from the use of chords in different classes. There are many guidelines for harmonic progression from various harmony texts [Butterworth, 1994; Gauldin, 1997; Piston, 1982; Schoenberg, 1990; Turek, 1988; Wittlich and Martin, 1989]. It could be generalised that in principle, the music moves in TDT or TSDT pattern. Harmonic progression in Bach's chorales can be understood from this view. The details of harmonic progression in Bach's chorales are discussed later in this section.

Regarding modulation, Bach's chorales are freely modulated. Each chorale has at least one modulation in it. The modulation in Bach's chorales is normally to a closely related key, that is from the original minor key to its relative major, or from the original major key to its relative minor, or from the original key to the new key with only one sharp or flat different from the original key. The modulation in Bach's chorales is normally obtained by means of a pivot chord modulation or a phrase modulation.

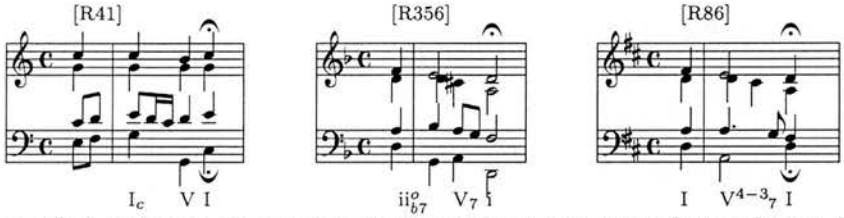
4.2.4.3 Cadence

A cadence concludes a musical phrase. The complete sonority in the ending chord is normally desired in the cadence (i.e. a complete chord in root position), especially when a feeling of firm finality is required. Each phrase normally ends with a pause chord (i.e. all voices come to a pause at the same time), but there are always exception in Bach's chorales. Chorales R141, R205, R207, R279 show examples of cadence where not all voices come to an end at the same time, but this is quite rare. In Bach's chorales, certain cadence forms frequently appear. Ebcioglu [1987] summarises 9 cadence cliches commonly found in Bach chorales in his report. We include this information in appendix C.

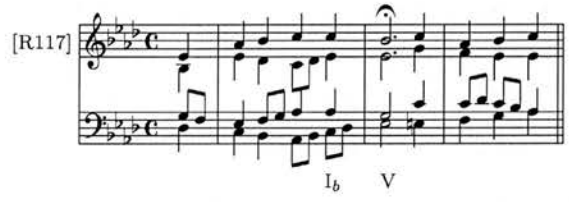
The cadences are generally grouped into four main categories: perfect cadence, imperfect cadence, plagal cadence and interrupted cadence. Boyd [1967] intensively studies the cadences in Bach's chorales, and makes a very useful summary of the occurrences of different cadence types. We reproduce Boyd's summary below:

Cadence Type	Root pos.	Inverted	Total	Approx %
Perfect cadence	1,241	211	1,452	73.0
Imperfect cadence	225	190	415	21.0
Plagal cadence	30	14	44	2.0
Interrupted cadence	33	nil	33	1.5
Others including cadences of modal chorales			50	2.5

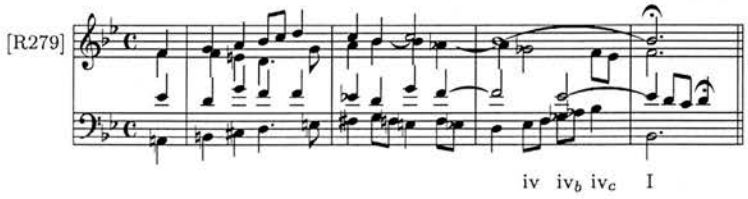
- Perfect cadence: It is the most common of all for its finality quality. As a rule, the piece ends with a perfect cadence with a major chord in root position. Boyd observes that inversions in cadences are common on the first of the two-chord cadence patterns. Of over nineteen hundred cadences he has analysed, only nineteen have inversions on the pause chords. The common perfect cadence formulae are I_c–$V_{(7)}$–I; ii_b–$V_{(7)}$–I; or $V_{(7)}$–I with 4-3 suspension at chord V.



- Imperfect cadence: Every chorale has at least one imperfect cadence. It usually appears during the first two phrases. The imperfect cadence may resolve to the tonic in the next phrase or may not resolve.



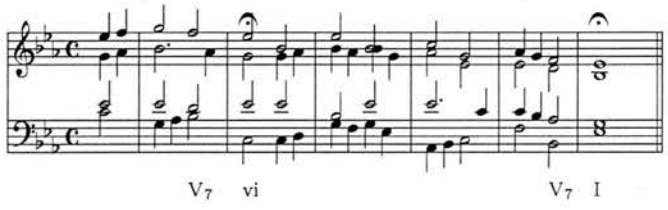
- Plagal cadence: This form of cadence is rare in Bach's style. Bach shows a very impressive application of this cadence type in chorale R279 [Butterworth, 1994].



- Interrupted cadence: This form of cadence is also rare. Butterworth [1994] and Boyd [1967] point out that it normally occurs as a penultimate cadence when the

melody can also be harmonised as a perfect cadence in the home key. Its use is to avoid consecutive perfect cadences in the home key.

[R179]



4.3 Implemented Domain Knowledge

We discussed the two dimensions of domain knowledge (i.e. musical structure and musical process) in chapter 2. In this section, we describe the domain knowledge implemented in our system in the same style.

4.3.1 Knowledge source

To model the behaviours of Bach's chorales is a great challenge. One of the important tasks is the knowledge elicitation task. Baroni and Jacoboni [1978] carried out an extensive study on the first two phrases of Bach's chorale melodies and concluded over 50 rules regarding the behaviours of chorale melodies from their study. The approach taken by Baroni and Jacoboni reveals the flavour of knowledge elicitation for a specific musical skill in which music theory tends to offer a more generalised version of the knowledge. We believe the same approach has to be taken for the construction of an effective knowledge base. However, our work focuses on control and not the basic knowledge of harmonisation. From the literature, the CHORAL project provides detailed rules employed in its system. There are about 350 rules in the CHORAL system, therefore, we decide to incorporate both traditional harmony theory and some of the knowledge employed in the CHORAL work [Ebcioglu, 1987]. We rewrite the knowledge content of the fill-in process in the CHORAL system and employ this knowledge in our system.

4.3.2 Theory of a four-part writing process

The four-part writing process involves a huge amount of knowledge of which human experts might not be aware. This dimension of knowledge is usually implicit in music theory. Looking at Bach's chorales, we can realise and associate musical materials

to music theory, but there is no way to associate the music with the way they are created. The implicit part of knowledge is very important in modelling four-part writing behaviours in machines. It does not only reveal the mental process but also provides an effective means of control knowledge for pruning the search space. There are not many studies of Bach's four part writing process in literature. It could be due to the lack of appropriate materials. Marshall studies composition process of Bach's vocal works. He comments that:

> "It is in the autograph scores of the four-part chorales, which in general contain only a few scattered corrections, that a classification of fair copy, revision copy, or composing score proves particularly elusive. The simple chorale setting seems as a rule to have presented Bach with remarkably few difficulties." [Marshall, 1972, p 69]

This leaves even fewer clues in the study of Bach's four-part writing process. However, the study does reveal the likely order of events in the process. Marshall suggests that Bach first writes down the entire melody, then the entire bass line and then the inner voices. But whether or not Bach writes the bass line down after each phrase of melody is written is not clear from the autograph scores. We also note that these observed processes may reflect the mental processes but these observations are in an extremely crude form.

Many texts also provide guidelines for a four-part writing process. Generally, the guidelines are clear at a local level, for example, how to chose a chord from a given pitch, how to connect between two chords, how to write a perfect cadence, etc. The knowledge becomes less clear when addressing the question of in which context, we use this particular chord and in which context we connect chords in a particular way. It becomes less clear because the application of the knowledge is context dependent. Therefore, it is natural that the knowledge is put down as guidelines and suggestions rather than a concrete theory.

4.3.2.1 Outlines of musical process

Although we cannot argue for detailed mental processes of a four-part writing task from our present knowledge, we can still construct a top level model of a four part writing process. We argue that the following top level processes are universal and proven to be an effective common practice (similar suggestions can be observed in various textbooks e.g. [Butterworth, 1994; Gauldin, 1997; Piston, 1982; Schoenberg, 1990; Turek, 1988;

Wittlich and Martin, 1989]). The modelling of the harmonisation of a given input melody may be described with the following strategies:

- Appreciate the overall form and structure of the piece.
- Outline the harmonic structure of the piece (i.e. harmonic progression).
- Outline the bass and other voices.
- Fill in all other details and stylistic touches.

In this dissertation, the *musical process* (e.g. compositional process, harmonisation process) may be thought of as defined by *domain dependent control knowledge.* We use this dimension of knowledge to control the way the search space is traversed. The details of the above control strategies are further refined to a finer grain size. Generally, we stop thinking about control when we do not want to control, or we do not know what parameters are to be controlled. The control knowledge may not be very clear when we discuss the domain knowledge in this section. However, more details are discussed in chapter 6.

4.3.3 The System Domain Knowledge

In this section, we describe our implemented domain knowledge in detail. The domain knowledge is declaratively described under three headings of the steps taken:

- Outlining the harmonic progression
- Outlining the bass line and the inner voices
- Filling in all other details and stylistic touches

The knowledge content in the first two steps are therefore from various sources (e.g. music theory, the CHORAL system, experts' empirical knowledge, etc.). The knowledge content of the fill-in process in the CHORAL system is employed in the last step.

4.3.4 Outlining the harmonic progression

We have presented the notation system employed in describing our musical domain at the beginning of this chapter. We have also presented general characteristics of the chorales. In our system, we start by analysing the input melody and grouping the melody into phrases. The information regarding the length of each phrase is pre-input by users.

We shall begin by introducing basic music vocabulary used in our system. Section 4.3.4.1 and 4.3.4.2 serve this purpose. Our chord vocabulary is slightly different from the CHORAL system as we do not have the notion of passing chord. The CHORAL has supertonic passing and subdominant passing chord. We also include altered chords such as secondary dominant, neapolitan sixth and augmented sixth in our chord vocabulary. Our system also works with original keys instead of transposing the melodies into the key of C major or the key of A minor.

4.3.4.1 Harmonic vocabulary:

- Knowledge of chords: the following triads, seventh chords and their inversions are allowed in the system:

Major mode	Minor mode
I	i
ii, ii_7	ii, ii^o, $ii^{\varnothing}_7$
iii	III, III^+
IV	iv, iv_7
IV_7	IV, IV_7
V, V_7	V, V_7, v
vi, vi_7	VI
♮VII, vii^o	vii^o, $vii^{\varnothing}_7$

  Apart from these basic triad and seventh chords, the following altered chords are also part of the chord vocabulary: V/V, vii^o/V, ♭II and augmented sixth chords.

- Knowledge of tonalities: Our system knows about tonality in both major and minor mode (melodic minor, harmonic minor and natural minor).

Major and Relative minor	Major and Relative minor
C major–A minor	F major–D minor
B♭ major–G minor	E♭ major–C minor
A♭ major–F minor	D♭ major–B♭ minor
G♭ major–Eb minor	F♯ major–D♯ minor
B major–G♯ minor	E major–C♯ minor
A major–F♯ minor	D major–B minor
G major–E minor	

4.3.4.2 Interval vocabulary:

- Knowledge of interval spelling: In traditional harmony, intervals of four semitones may carry different tonal connotations if one is a major third interval and the other is a diminished fourth interval. There is an ambiguity when determining types of the intervals from the interval size alone (see table below). Our system overcomes this limitation by using inteval spellings, for examples, an interval between $\hat{c}\sharp$ and $\hat{e}\sharp$ will be read as a major third and an interval between $\hat{c}\sharp$ and $\hat{f}$ will be read as a diminished fourth.

Spellings	semitones	Spellings	semitones
diminished unison	-1	unison	0
augmented unison	1	diminished second	0
minor second	1	major second	2
augmented second	3	diminished third	2
minor third	3	major third	4
augmented third	5	diminished fourth	4
perfect fourth	5	augmented fourth	6
diminished fifth	6	perfect fifth	7
augmented fifth	8	diminished sixth	7
minor sixth	8	major sixth	9
augmented sixth	10	diminished seventh	9
minor seventh	10	major seventh	11
augmented seventh	12	diminished octave	11
octave	12		

4.3.4.3 Determine the chord from an input melody

In the beginning, our score has information of an input melody and the length of each phrase. For convenience, the input melody line is grouped into phrases with beat information. For example, two minims in a $\frac{4}{4}$ time signature will be grouped into four beats. The first minim covers beat one and two. The second minim covers beat three and four. The soprano pitch in each beat is used to determine plausible chords. The bullet points below summarise how plausible chords are determined.

- Determine the soprano pitch to be used: By default the pitch at the beginning of the beat is used to find the chord. If there are more than one pitch in a crotchet beat, other pitches will be tried later.

- Determine the tonality at that point: The input melody (soprano) is notated with references to the original key signature of the chorale. The system also keeps track of the current tonality which may not be the home key. This information is employed to find a plausible chord for a given input melody.
- Determine the chord to fit the harmonic rhythm: The harmonic rhythm in chorales is set to change in every crotchet.
- We have decided to embed the control for chord choices at this stage at the object-level. For example, if the melody is ĉ and the key signature is F major then the plausible chords could be C, F, C_7, Am, etc. However, users cannot explicitly specify the order of how these chords should be generated with the control language at this level.

4.3.4.4 Chordal textures, Doublings and Inversions

- The final texture (i.e. chordal texture and rhythmic texture) is determined during the detailed decorations at the final stage. At the current stage, the chordal texture is a triad or a seventh chord. The open position or close position texture of the chord is indirectly conditioned by the melodic shape.
- In a four part writing, if the chord is a triad, one of the pitches in the triad will be doubled. The following standard part writing guidelines are employed:

Chord types	Heuristic
major chords	doubling root, fifth
minor chords	doubling root, third, fifth
augmented chords	doubling root, third, fifth
diminished chords	doubling third, fifth
cadential $\frac{6}{4}$ chords	doubling fifth
any chord types	no doubling of leading notes

- Chords could be used in their root position forms or inversion forms. The root position and the second inversion are freely used by major, major-seventh, minor and minor-seventh chords. We restrict the second inversion for a cadential $\frac{6}{4}$ context. Diminished chords are used in the first inversion or root position in a $\text{vii}^o \rightarrow \text{I}$ progression.
- An incomplete chord is also allowed in the final decoration stage. As a general rule, omit the fifth and add an extra root or third.

4.3.4.5 Phrase beginnings

The first phrase always begins in the home key, however, other phrases may begin in other tonal areas. Generally speaking, a phrase starts with a chord or chords of tonic functionality.

- A phrase with a down-beat (thesis) opening starts with a tonic chord.
- A phrase with an up-beat (anacrusis) opening starts with a tonic chord, or a subdominant or a dominant chord which moves to a tonic functional chord on the strong beat.

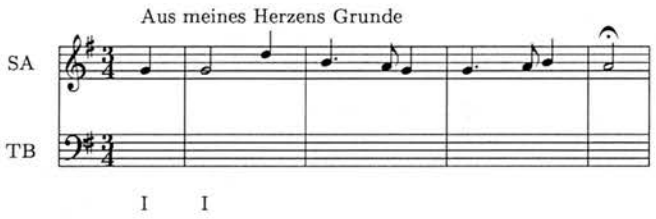


Figure 4.2: Phrase beginning

Figure 4.2 shows outline chords at the beginning of the first phrase of chorale 'Aus meines Herzens Grunde'. The system decides to use tonic chord in both location. At this stage, only the chord names are outlined, their voicings, inversion types or doublings are not yet determined.

4.3.4.6 Chord transition from left to right in major and minor mode

The chord transition knowledge below is adapted from Ebcioglu's CHORAL, see Ebcioglu [1987, p 240–242]. This progression is used in the body part (not the beginning and the ending of the phrase). The notation A $\rightarrow$ B, C, D reads as chords B, C, D are the plausible choices to follow chord A. The knowledge expressed here is not in declarative form. In this example, chord B will be tried before chord C and D respectively.

- Summary of chord transition from left to right in a major mode:
  - no repeat of cadential $\frac{6}{4}$ chord (I_4^6)
  - I $\rightarrow$ ii, ii_7, IV, vi, vi_7, V, V_7, vii^o
  - ii, ii_7 $\rightarrow$ V, V_7
  - ii, ii_7 $\rightarrow$ I

- ii, ii_7 → vi, vi_7 in root position
- iii → IV, vi, vi_7
- IV, IV_7 → ii, ii_7, V, V_7, vii^o
- IV → I, IV_7
- V, V_7 → vi, ii, IV, iii, vii^o, I
- vi, vi_7 → IV, ii, V, V_7, iii
- vii^o → I
- ii, IV → I_c → V, V_7
- The following chords: I, ii, ii_7, IV, V, V_7, vi, vi_7 may repeat

- Summary of chord transition from left to right in a minor mode:
  - i → ii^o, $ii^{\varnothing}_7$, ii, iv, iv_7
  - i → IV, IV_7, V, V_7, v, VI
  - i → vii^o, $vii^{\varnothing}_7$
  - ii^o,$ii^{\varnothing}_7$ → V, V_7
  - ii → i if some voice (not the bass voice) rises from the 5th of ii ($\flat\hat{6}$) to the root of i ($\hat{1}$)
  - ii^o, $ii^{\varnothing}_7$ → i
  - III^+ → i, VI
  - iv, iv_7 → ii^o, $ii^{\varnothing}_7$, i, V, V_7
  - iv, iv_7 → III^+, vii^o, $vii^{\varnothing}_7$
  - IV, IV_7 → V, V_7
  - V, V_7 → VI, vii^o, $vii^{\varnothing}_7$, i
  - v → iv, iv_7, VI
  - VI → iv, iv_7, ii^o, $ii^{\varnothing}_7$, V, V_7
  - vii^o, $vii^{\varnothing}_7$ → i
  - ii^o, $ii^{\varnothing}_7$ → I_c → V, V_7
  - iv, iv_7 → I_c → V, V_7
  - VI → I_c → V, V_7
  - The following chords: i, III, iv, iv_7, V, V_7, VI, VII may repeat

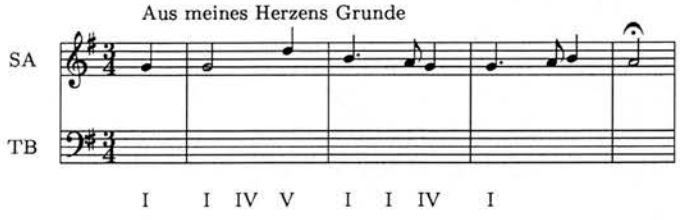


Figure 4.3: A plausible harmonic progression

Using the same melody as a running example, the outlined harmony in figure 4.3 is one of many plausible solutions.

4.3.4.7 Phrase endings

Each phrase is ended with a cadence pattern in some key. The cadence patterns illustrated in figure 4.4 and 4.5 are adapted from the cadence constraints in the CHORAL system. The table summarises the determination of all cadence types used in our system.

- **Major mode**: The following cadence patterns in figure 4.4 are used in the major mode. The dominant seventh (V_7), diminished seventh ($ii^{\varnothing}_7$; $\sharp iv^{\varnothing}_7$; $vii^{\varnothing}_7$) and altered form chords (e.g. tierce de picardie) are also possible alternatives when appropriate. When the inversion is specified (e.g. V_a), the chord must be in that specified position.

Ending degree	Plausible Cadence patterns	Key
$\hat{1}$	V-I, vii^o-I, V_a-vi	Tonic
$\hat{1}$	V-I	Subdominant
$\hat{2}$	I-V, ii-V, IV-V	Tonic
$\hat{2}$	V-I, vii^o-I	Dominant
$\hat{2}$	V-I	Relative-minor of Subdominant
$\hat{3}$	V-I, vii^o-I	Tonic
$\hat{3}$	i-V, ii-V, IV-V	Relative-minor of Subdominant
$\hat{4}$	V-I, vii^o-I, V_a-vi	Subdominant
$\hat{4}$	V-I	Relative-minor of Subdominant
$\hat{5}$	I-V, ii-V, IV-V, IV-I	Tonic
$\hat{5}$	V-I, vii^o-I	Dominant
$\hat{5}$	V-I, vii^o-I	Relative-minor of Dominant
$\hat{6}$	V-I, V_a-VI	Relative-minor
$\hat{6}$	V-I	Subdominant
$\hat{7}$	I-V, ii-V, IV-V	Tonic
$\hat{7}$	V-I, vii^o-I	Dominant
$\hat{7}$	i-V, ii-V, IV-V	Relative-minor

Figure 4.4: Cadence patterns in major modes

- **Minor mode**: The following cadence patterns in figure 4.5 are used in the minor mode. The dominant seventh (V_7), diminished seventh ($ii^{\varnothing}_7$; $\sharp iv^{\varnothing}_7$; $vii^{\varnothing}_7$) and altered forms chords (e.g. tierce de picardie) are also possible alternatives when appropriate. When the inversion is specified (e.g. V_a), the chord must be in that

specified position.

Ending degree	Plausible Cadence patterns	Key
$\hat{1}$	V-I, vii^o-I, V$_a$-VI	Tonic
$\hat{1}$	V-i, vii^o-i	Tonic
$\hat{2}$	i-V, ii^o-V, iv-V	Tonic
$\hat{2}$	V-I	Relative-major of Dominant
$\flat\hat{3}$	V-I, vii^o-I, V-i, vii^o-i	Tonic
$\flat\hat{3}$	V-I, V$_a$-vi	Relative-major
$\hat{4}$	V-I, V-i	Subdominant
$\hat{4}$	V-I, vii^o-I	Relative-major of Dominant
$\hat{5}$	i-V, ii^o-V, iv-V	Tonic
$\hat{5}$	V-I, vii^o-I, V-i, vii^o-i	Dominant
$\hat{5}$	i-V	Subdominant
$\hat{5}$	V-I	Relative-major
$\flat\hat{6}$	V-I, V$_a$-vi	Relative-major of Subdominant
$\flat\hat{7}$	V-I, vii^o-I	Relative-major of Dominant
$\flat\hat{7}$	V-I, vii^o-I, V-i, vii^o-i	Dominant
$\hat{7}$	i-V, ii^o-V, iv-V	Tonic

Figure 4.5: Cadence patterns in minor modes

- The perfect cadence in both major and minor modes may be elaborated with a cadential $\frac{6}{4}$ chord. The cadential $\frac{6}{4}$ chord must follow a supertonic or a subdominant chord in a major key; and must follow a supertonic, a subdominant or a submediant chord in a minor key.

The CHORAL system also uses predefined cadence patterns as cadence cliches. The cadence cliches also dictate the way voices are filled. In the CHORAL system the selection of cliches depends a lot on voicings preceding the cadence. In our system, we take a different approach, the determination of harmony and voicing are seen as two separate tasks. Further more, the decision on the cadence pattern, and voicings can be done in any preferable order (e.g. before the body of the phrase is filled or after). Returning to the running example (see figure 4.6), the system decides to use an imperfect cadence (I-V) for this phrase. The selection of cadence is based on the tables in figure 4.4.

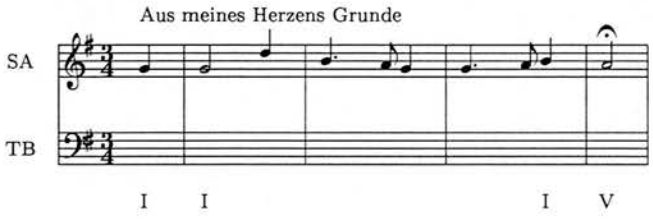


Figure 4.6: Phrase ending

4.3.4.8 Modulation procedures

The chorale is allowed to modulate freely to its related key. The pivot chord modulation tactic used here is adapted from the CHORAL system, see Ebcioglu [1987, p 242–245].

	Chords in new key	
Entry Type	Pivot chords	Entry chords
Tonic–Dominant	I	V, V_7
	i	V, V_7
Supertonic–Dominant	ii^o, ii^o_7	V,V_7
	ii, ii_7	V, V_7
Mediant–Dominant	III	V,V_7
Subdominant–Dominant	iv, iv_7	V, V_7
	IV	V, V_7
	$\sharp iv^o$, $\sharp iv^o_7$	V, V_7
	$\sharp iv^o_7$	V,V_7
Dominant–Dominant	V	V, V_7
Submediant–Dominant	VI	V, V_7
Leading–Dominant	vi, vi_7	V, V_7
	VII	V,V_7
Tonic–Leading	i	vii^o, vii^o_7
	I	vii^o
Mediant–Leading	III	vii^o
Subdominant–Leading	iv, iv_7	vii^o, vii^o_7
	IV	vii^o
Submediant–Leading	VI	vii^o, vii^o_7
	vi, vi_7	vii^o
Leading–Leading	$\flat$VII	vii^o

The modulation rules in the CHORAL system are more complete than the version we implement. However, the simple pivot chord modulation in the table above is powerful

enough to lead the tonality to any related key. The modulation mechanism in the CHORAL system is driven by accidentals which are generated by the chord skeleton view (i.e. alien accidentals pull the tonality of the chorale). In our implementation, the modulation is determined with the harmonic progression plan. We believe our technique is better since the modulation pattern and the overall harmonic plan can be designed in a global view (e.g. modulation is encouraged in the middle part of the chorale piece even though there is no accidental in the melody at all). The modulation

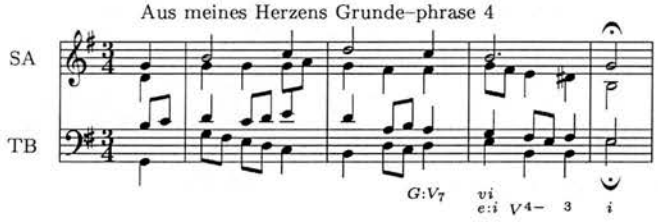


Figure 4.7: A modulation passage

example in figure 4.7 is taken from the fourth phrase of the same chorale. The chorale is modulated to the key of E minor which is the relative minor of G major.

4.3.4.9 Harmonic characters of chorales

- Harmonic progression in each phrase is reasoned in the same fashion at the local level (chord level). However, the system also monitors the harmonic progression of each phrase at a global level to maintain the integrity of the whole.
- A phrase starts with a tonic functional chord in some keys.
- A phrase ends with some cadence patterns in some keys.
- The common perfect cadence formulae are I_c–$V_{(7)}$–I; ii_b–$V_{(7)}$–I; or $V_{(7)}$–I with 4-3 suspension at chord V.
- The first phrase starts in the home key.
- The penultimate phrase should end with an interrupted cadence in home key, if possible.
- The last phrase ends with a perfect cadence in the home key.
- Two consecutive phrases cannot end with the same cadence type in the same key.
- Harmonic progression in each phrase must be in different patterns.
- Harmonic progression in the body of a chorale should progress away from the home key and must get back to the home key in the final cadence.

- Modulations are freely allowed in a body of a chorale; closely related keys would be tried first (In this implementation, the modulations are only to closely related key).

4.3.5 Outlining the bass line and other inner voices

Our system allows all voices to be outlined with flexibility. For example: all voices may be outlined from left to right in a chord by chord fashion; each voice can be outlined in a voice by voice fashion and in any order; harmonisation can be done in any phrases in any order. However, the atomic control definitions implemented at the current stage only support the harmonisation process as described earlier (e.g. determined harmonic progressions, outline the bass line, outline the inner voices and then fill-in detail decorations). Standard voice leading practices are given below:

4.3.5.1 Standard voice leading procedures

- Voice range: This is the standard range of each voice in our system, see page 67.
- Diminished or augmented progression intervals are not allowed.
- Parallel fifth and parallel octave progressions are not allowed between any two voices.
- A parallel progression from a perfect fifth to a diminished fifth interval is allowed when both voices move by step. The parallel progression from a diminished fifth to a perfect fifth is still considered as a parallel fifth and therefore forbidden.
- Hidden unison is allowed only between the tenor and the bass.
- Exposed octave is not allowed between the soprano and the bass except when the soprano moves by step.
- Part crossings are not allowed in our system.

4.3.5.2 Outline the bass voices

- Contrary motion between the bass line and the soprano line is desired when outlining the bass.
- As a general preference, the bass maintains the direction from the start to the cadence point in each phrase.
- Diminished or augmented progression intervals are not allowed. A skip larger than a fifth is not allowed, but an octave skip is possible.

- The chorales are characterised by the ascending bass line towards a cadence point. This feature should be employed whenever possible.

The example in figure 4.8 shows the outline bass. At this stage, an inversion type of a chord is determined.

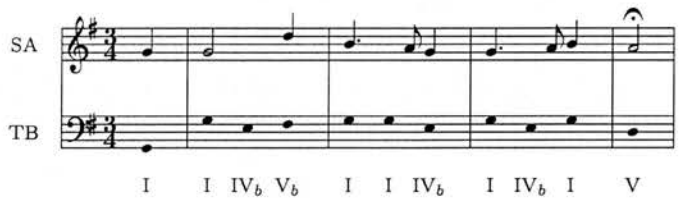


Figure 4.8: Outline bass

4.3.5.3 Outline the inner voices

- As a general preference, the alto and the tenor maintain their directions by moving in step or third in the same direction.
- Diminished or augmented progression intervals are not allowed. A skip larger than a fifth is not allowed.
- The alto and the tenor should be less active than the bass line.

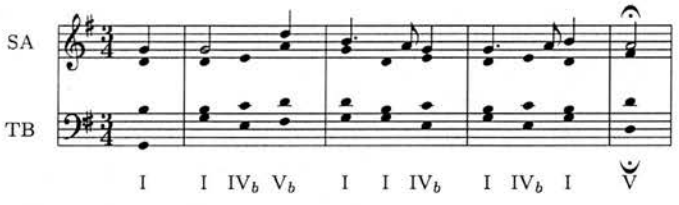


Figure 4.9: Outline voices

4.3.6 Fill in all other details and stylistic touches

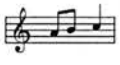
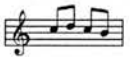
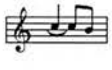
In this section, we summarise the implemented domain knowledge regarding the decoration of all voices. The knowledge content in this section is taken from the fill-in process of the CHORAL system (the full details of the CHORAL system are given in appendix C). The implementation details of this knowledge are in chapter 6.

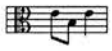
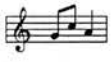
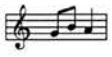
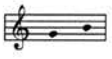
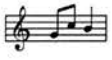
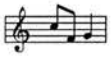
In brief, the problem solving process in the CHORAL system is carried out in the following steps: the chord skeleton view; and the fill-in process (which embeds the fill-in view, the time slice view, the melodic string view and the merged melodic string view into one process). The fill-in process takes a chord produced by the chord skeleton view and decides on the voicing of that chord. The fill in process also performs the decoration of the final result with the fill in of suspensions, passing notes, neighbor notes, etc. The CHORAL system organises the knowledge in this section in terms of the generate section, the constraint section, and the heuristics section.

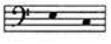
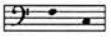
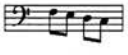
In our system, the outline pitch is equivalent to pitches in the chord skeleton view. The fill-in of detail decorations works with these outlined pitches. The behaviour of the generation section in the fill-in process is classified (follow the CHORAL) using three main states.

4.3.6.1 A normal state

In this state, the outline pitches may be decorated with passing notes, neighbor notes, etc. The table below summarises plausible decoration in this state.

Outlined voices	Possible decoration	Remarks
		In a simple form, the outline pitches of any voices are filled as crotchets.
		When an interval between the two pitches (any voices) is unison, the following decorations are plausible. From top down: a decoration with a third up and a descending accented passing note; a neighbor note; a neighbor suspension note.
		
		
		A passing note may be filled between an interval of third, either up or down.
		
		When an interval between the two pitches (any voices) is a descending second, a neighbor descending passing note or a suspension may be tried.
		

Outlined Alto-Tenor	Possible decoration	Remarks
		When an interval between the alto or the tenor voice is unison, a skip down fourth may be tried.
	 	An ascending third or fourth skip may decorate an ascending second outline pitches. The same is true for the descending second outline pitches.
	  	An escape note may decorate an interval greater than a second but less than seventh in any direction. In the case of an ascending direction, a suspension may be introduced as well.

Outlined Bass	Possible decoration	Remarks
		When the two bass pitches are in step movement, if the first bass is not in a root position, then a third skip downward may be tried.
 	 	An accented passing note may be filled in between interval of a third or a fourth in the bass.

The fill-in starts with a normal state, the state may change to suspension state or descending passing state during the transition (see details in appendix C). Note that the second crotchet after the transition is the outline pitch for the next transition.

4.3.6.2 A suspension state

In this state, the previous outline pitch is to be held over the stronger quaver. The following are plausible choices after the resolution.

Outlined voices	Possible decoration	Remarks
		In a simple form, the resolution is resolved on the weaker quaver
	 	If the original outline pitches are the same pitch, then the suspension may be held for a crotchet value or decorated with a step down below the resolution.
		When the two outline pitches form a descending second, a suspension may be tried.
		When the two outline pitches form a descending third, a descending accented passing note may be tried.

4.3.6.3 A descending passing note state

This informs the control that the current outline pitch is to be filled in the weaker beat follows the accented passing note. The following are plausible choices after the resolution.

Outlined voices	Possible decoration	Remarks
		In a simple form, the resolution is resolved on the weaker quaver
		When the two outline pitches form a descending second, a suspension may be tried.
		When the two outline pitches form a descending third, a descending accented passing note may be tried.

4.3.6.4 The Constraints and Heuristics sections in the Fill-in process

The constraints employed in the fill-in process are associated with the sub-view in the process (i.e. fill-in view, melodic string view, merged melodic string view and time slice view). We implement constraints in the CHORAL as our atomic test definitions and implement heuristics as our atomic measure definitions. The details of knowledge in this section are clearly explained in appendix C. Now, let's return to the running example where we fill in the final decorations from left to right. The first example in figure 4.10 shows an early stage of the fill-in process and the second example shows a complete solution.

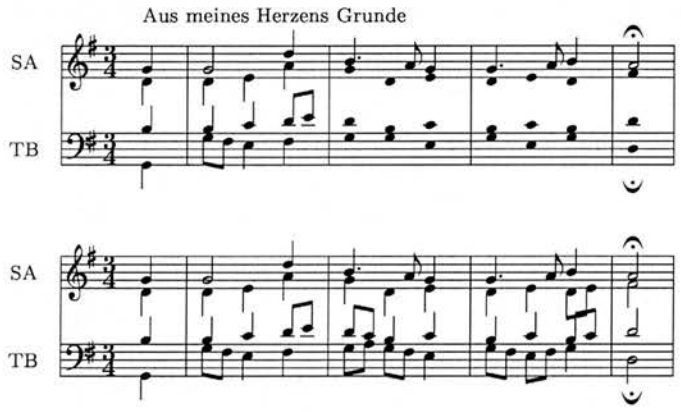


Figure 4.10: Fill in decorations

4.4 Summary

This chapter summarises the domain knowledge in our system. We describe the domain knowledge loosely under the top level chorale harmonisation process: outline the harmonic structure of the piece; outline the bass and other voices and fill in all other details and stylistic touches. In the following chapters, we will explain our representation framework for the proposed musical structures and musical process. A more complete picture of the control in our explicitly structured model is given in chapter 6.

Chapter 5

Musical Structure Representations

Introduction

In this chapter, we discuss the issue of knowledge representation in detail. As a running example, we start with the melody of Bach's chorale entitled *Aus meines Herzens Grunde* (see figure 5.1). Our system aims to provide a four-part harmonisation to the given melody. First of all, we need a representation system which is expressive and which facilitates construction of a reasoning system.

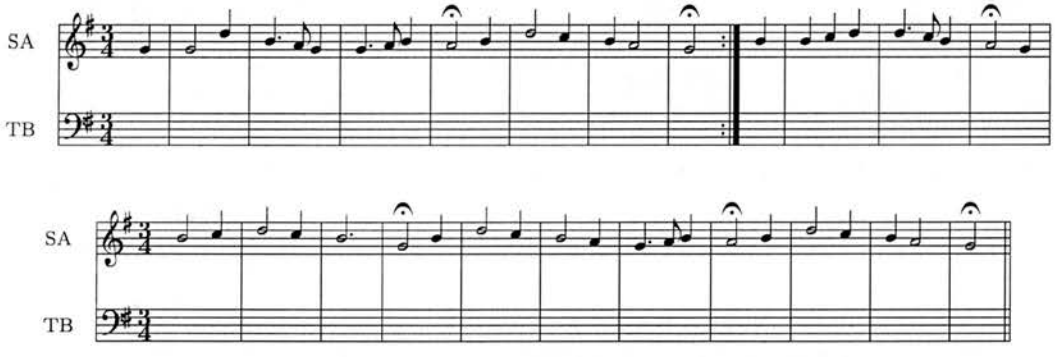


Figure 5.1: Bach's Chorale melody—Aus meines Herzens Grunde

Generally, we solve problems by first analysing the problem. Then we devise a plan which will later be performed. The process is repeated until one is satisfied with the result. Our task in this chapter is to devise an appropriate knowledge representation system which will facilitate our problem solving strategies. First, we must understand

the nature of the problem. Let us consider the following problem solving strategy:

Understand the overall structure of the piece:

- Time signatures, key signatures.
- Form of the pieces, how many phrases, where are the repetitions?
- The character of the melody line, implied harmony.
- Suitable choices for melody, harmony and texture for the genre.

Initiate a plausible strategy from the existing knowledge:

- Decide on harmonic rhythm, harmonic progression.
- Decide on cadence types, texture, stylistic points.
- Decide on order of activities, what to do first, next and last?

Try out the ideas:

- Fill in sections according to a plan.
- Look for alternative approaches if the plan is not feasible.

Analysis of what has been done:

- After pieces of information are filled in, they may lead to some better ideas.
- The process continues until the work is accepted.

The knowledge perspectives in terms of musical structures and compositional processes are observable in the outline above. The compositional process is not merely a procedural representation of the domain knowledge. It is the knowledge that allows us to come up with the outline as such. It captures the implicit part of the knowledge which is not apparent from a finished product. We argue that the compositional process provides a vital ingredient in shaping the search strategy (i.e. a reduced search space at meta-level, a traversing path for a partial search). This issue will be elaborated in the chapter 6.

5.1 Domain Knowledge Representations

The strategic outline above reveals one way of looking at and solving the harmonisation problem in many small steps. This tactic is known as a *problem reduction* technique [Nilsson, 1971; Stacy and Charles, 1992]. The problem reduction technique is an effective problem solving tactic. However, there is a problem associated with it. When we break a knowledge chunk into many smaller chunks, the dependencies among these smaller chunks may appear. For example, within the genre of four-part harmonised

chorales, we may attempt to encode all the cadence patterns found in Bach's chorales into the system. This is possible but is a very tedious task. To build further knowledge on top of this huge data base seems to increase the amount of effort required exponentially. We could remedy this by reducing the size of the knowledge base. This can be done by generalising the domain into a generalised concept, in this case, a skeleton of cadence patterns. Trying to construct a cadence by approaching from a skeleton and then filling in the details divides the problem into two dependent main steps. The dependencies may be local or global and they play an important role in the search control issue. In the overall picture, the choice of knowledge representation in the system has a lot of consequences for the structure of the system.

We argue that the representation of the domain knowledge must accommodate:

- The representation of musical structures where the system should be able to talk about the musical structure knowledge in music (e.g. pitch, melody, harmony, texture, form, cadence)
- The representation of a harmonisation process where the system should be able to talk about compositional process knowledge (e.g. when writing a cadence, what is the musical thinking applied by the composers).

We also argue that the basic building blocks created from observable properties (within a scope of conventional musical notation system) are expressive enough to allow the system to refer to and talk about musical knowledge. There are many plausible formats to represent music with the two dimensions of pitch and time. For example, a pitch may be encoded in terms of *Height, Chroma, Position in the circle of fifths* or *Name, Register* or *Key positions* in a piano while time may be encoded in terms of *physical time, duration, beat* or *rhythm.* There are many alternatives available. The main concern at this stage is that the chosen primitives and representation language must be expressive enough to allow discussions about the application domain. In our musical domain, we list some plausible primitives below.

- Pitch related attributes (e.g. Name, Accidental, Register, Frequency)
- Sonority related attributes (e.g. Timbre, Intervals, Chords)
- Performing instruction related attributes (e.g. Forte, Piano, Accent)
- Structure related attributes (e.g. Key, Metrical structure, Phrases, Cadences)
- Process related attributes (e.g. Strategies, Order of tasks)
- General information attributes (e.g. Genre, Piece name)

- Physical time (e.g. Clock time)
- Notated musical time (e.g. Pitch duration, tempo, beat)

We construct knowledge of musical structure based on these primitive propositions *(e.g. pitch name, pitch duration, pitch register, time, etc).* On the other hand, we construct compositional process from the manipulations of these primitive propositions *(e.g. subtract pitch, add pitch, etc).* Both musical structure and compositional process are hierarchical by nature. They may be constructed from many lower level structures/processes. Before going any further, we clarify the terminology used in the discussion.

- Pitch: An abstract representation of a set of tones. At this level the distinctions between pitches are in their highness and lowness which could be referred to as different frequencies or names. In our implementation, we represent a pitch as a member of {1 2 3 4 5 6 7} × {*nat* ♯ ♭ ♭♭ x} × Integer [Harris *et al.*, 1991].
- Notated musical time: Time representation used to quantify the length of pitches (e.g. duration) or other properties (e.g. beat). In our implementation, time is represented using integers. The smallest unit of 1 is equivalent to a demisemiquaver (thirty-second note).
- Note event: An event constructed from pitch, onset time and duration time.
- Musical event: An abstract representation for any properties observable within a scope of conventional musical notation system.

5.2 Representation of Musical Structures

We see the representation of musical structures as the fundamental ground work interwoven with the representation of compositional processes. The primitives of compositional processes are dependent on the primitives of musical structures. However, in this section, we concentrate only on the musical structure issue. In this thesis, the concepts of the musical structure are constructed from pitch and time attributes using a logical representation framework. We now discuss this in detail.

5.2.1 Score representation

We want the chosen attributes and their constructions to represent appropriate knowledge needed in our reasoning process. The most common notation representing musical

knowledge in western music is a notation system based on symbols including: a five-line stave, a key signature, a time signature, pitch symbols, etc.

For example: 

This notation system has been developed over a long period of time and has achieved a certain expressiveness. Western music theory has been developed around (and alongside) this representation for centuries. Therefore, we propose a representation paradigm based on the concepts of this notation. We name our representation: the *Score* representation. The score groups the domain knowledge under two main components:

- Musical materials;
- Interpretations of the musical materials.

The analogy for this paradigm is the mapping between the conventional musical notations (the musical materials which we choose to describe musical scores) and the mental world (interpretation or analysis of the materials according to our understanding). The musical materials section captures the domain in the concept of pitches and their associated properties (e.g. duration, time signature). The interpretation section captures an interpreter's beliefs and understandings of the domain in concepts of musical structures. These concepts are declarative relationships formed between different parts of the score (of whatever size). Different interpretations of the same material yield different viewpoints on the knowledge content. Generally, the richness, gained from the multiple views, comes with the complications of having to deal with a more complex truth maintenance process when the facts under the multiple structures change. We have not dealt with any truth maintenance in our system. However, we get around this problem by allowing the system the ability to present different views when required. In this case, the system has access to the views it is working on via the chronological backtracking mechanism in Prolog.

5.2.2 Musical materials

We have decided to construct the musical material part based on two main concepts: *Note events*, and *lines*. The note events is constructed from the pitch notation based on the conventional western notations and time. The pitch notation describes a pitch as it should be read from the stave. The time describes both the notated time and the

physical time. The musical material part of the score is required to, at least, capture all the information available from the conventional western music notations. It is possible, then, to use this information in further processing to obtain realised sound events, or to reason about other properties of the musical materials. We explain the concepts of note events and lines below.

5.2.2.1 Note events

The construction of the note events is decided to be from the pitch notation and the notated time: *Name, Accidental, Register* and *Onset time–Duration* [Wiggins *et al.*, 1989]. This is, of course, not a complete description of a musical event in all applications. Other information (e.g. performing instructions) may be incorporated later on for appropriate applications. This point will be clear in the discussion of the score notations in the latter part of this chapter. By constructing the note event from pitch related attributes and time, any operation on note events must also take the time dimension into account.

5.2.2.2 Lines

A line contains a list of events. Conceptually speaking, we think of a line as a list of events. In our current work, we represent four parts: soprano, alto, tenor and bass as four different lines forming from pitch events. In other applications, each line may represent each instrument, each player or each hand of a pianist. This is entirely up to the task one wishes to model.

5.2.3 Interpretations

All knowledge pertaining to the domain can be discussed at any level with reference to the appropriate materials (e.g. soprano pitch at bar 1, cadence at the end of the first phrase, harmonic rhythm before the final cadence). A predicate formed from a set of events portrays a certain concept. The concepts are hierarchical: we can construct our interpretations (i.e. concepts formed from various events) by hierarchically constructing these concepts. Meanings of events are expressed as *terms* definable over events using Prolog style terms (i.e. constants, variables and compound terms). Rowe [1988] classifies Prolog style compound terms into six categories: a type predicate, a property predicate, a relationship predicate, a database predicate, a function predicate and a

probability predicate. Further meanings are defined in a similar way using properties of events as well as previously defined terms. All types of Prolog style terms described above are common in our system. We give some examples in figure 5.2.

Categories	Examples	Meaning of the examples
Constant		
integer	34	An integer constant '34' (which could mean anything apart from the quantity 34)
float	0.6	A float constant '0.6'
atom	'C major'	An atomic constant 'C major' corresponds to a term used in music theory.
Variable		
variable	Cadence	A variable stands for some definite but unidentified object.
Compound Terms		
type	scale('C major',[c,d,e,f,g,a,b])	The scale of C major has the pitch class of {c d e f g a b c}.
property	chord(dominant,root(?Pitch))	A variable pitch is a root of a dominant chord.
relationship	triad_Cmajor(c,e,g)	Pitch class c, e, g form a C major triad.
database	chromaticInterval(min-third,3)	A database entry indicates a number of semi-tones and their equivalent name (a minor third interval).
function	gtpitch(+Pitch1,+Pitch2)	Return 'yes' if Pitch1 is higher than Pitch2 and return 'no' if Pitch2 is higher than Pitch1.
probability	perfectCadence(0.8)	There are 0.8 probability that the cadence is a perfect cadence.

Figure 5.2: Example of Terms used in the system

Music analysts generally describe musical structures in terms of: *form, melody, harmony, texture.* For convenience, the interpretations of the music materials in our system are based on the same view. In the chorale harmonisation application, the knowledge of music theory must cover the knowledge of pitch, mode, keys, triad chords, seventh chords, functional tonalities, modulations, simple ornamentations, and the

structure of chorales. Here, the interpretation part of the chorale is grouped into various objects (e.g. a chorale object, a phrase object). This allows a neat organisation of both general information (i.e. general concepts associated to the piece level in the chorale object) and specific information (i.e. local concepts in the phrase object).

Each object holds musical concepts in terms of form, melody, harmony and texture. The interpretation part must be expressive enough to allow discussions of music theory at the appropriate level of the task. We illustrate how the interpretations of the musical materials under various concepts look like. We will base the example on the chorale '*Aus meines Herzens Grunde*' (see figure 5.1 and 5.2):

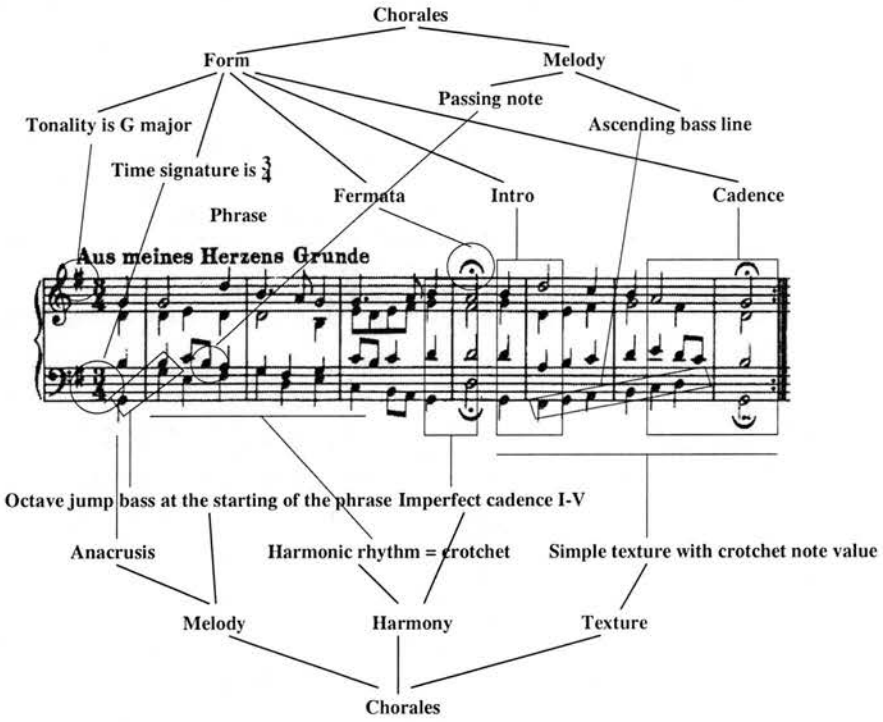


Figure 5.3: Representation of musical structures (see section 5.3.3 for rendition in our representation)

5.2.3.1 Form

The formal structure of works in a particular genre holds to the common patterns. In the case of Bach's four-part chorales, the bar form (discussed in chapter 4) pattern is

visible as a formal structure. Examples of the knowledge relevant to formal chorale structures are given below (based on the chorale '*Aus meines Herzens Grunde*', see figure 5.3).

- Time signature is $\frac{3}{4}$
- Key signature is G major.
- The chorale has six phrases.
- The number of beats in the phrases are 12–9–9–12–12–9.
- The layout of the chorale is A–B–B'–A'–A"–B with a repeat of the first two phrase (i.e. A–B–A–B–B'–A'–A"–B).

5.2.3.2 Melody

The contour of the given melody, its range, and its rhythmic shape are important features for the harmonisation tasks. The system uses this information in planning the melodic shapes and rhythmic patterns of all other voices. For example, the system analyses the melody input and has the following knowledge in its melody concept (based on the same chorale example):

- Similarity of melody in each phrase: For example, the second phrase and the last phrase share the same melody line.
- All phrases open with anacrusis.
- All phrase end on a strong metrical structure beat.
- The longest note value observed is a minim, the shortest one is a quaver.
- There is no accidental in the melody line.

5.2.3.3 Harmony

The implied harmony can be derived from the given melody. The system would employ the available knowledge to infer more new knowledge. In this process, the consistency of new knowledge must be maintained. In other words, each application of an inference rule must be considered not only with the local information (e.g. the melody and its adjacent neighbours) but also with other available information of the whole chorale, so that the locality in the selected inference rule will not conflict with other existing knowledge (we refer to this as the "context dependency problem"). Figure 5.3 illustrates the four-part harmonisation of the first two phrases. We expect the following knowledge in the interpretation part.

- The first phrase begins with a tonic chord.
- The first phrase ends with an imperfect cadence.
- The harmonic progression of the first phrase is: I–IV$_b$–V$_b$–I–V–vi–IV–V–I.
- The second phrase ends with a perfect cadence.

5.2.3.4 Texture

The desired texture is worked out from a given melody. We discussed the typical chorales texture in chapter 4. There are certain stylistic points that the system incorporates in the writing process.

- The decision to voice in a closed or opened position.
- The decision to voice in a note against note, or many notes fashions.
- The decision of the rhythmic patterns in each voice, to create variety.
- The decision of the final decoration, to make a coherent texture of the whole piece.

The system employs these concepts (i.e. form, melody, harmony, texture) in the harmonisation task. During the harmonisation process, more facts are added to the interpretation part of the score to make the knowledge base more complete. The derived knowledge is non-volatile (i.e. new knowledge is added to be part of a score but it is dependent on the backtracking mechanism in prolog).

5.3 Score Notation

Our knowledge representation is based on a logical approach. The knowledge of the world is represented as an atomic sentence which may be constructed with appropriate logical connectors. In brief, we may write our knowledge in a Horn clause: $A \leftarrow L_1 \wedge L_2 \wedge ... \wedge L_n$ where L is a literal which is an atomic sentence 'A'.

In our implementation, we use the term *Score* to denote the representation of the musical knowledge. The score represents two main components: musical materials and interpretations. We notate the score using a constructor:

score(Musical materials, Interpretations)

As long as the musical material part captures enough information (i.e. we are satisfied with the fidelity at the conventional musical notation level) and if we possess the right

transformation knowledge, then the original musical materials can be shaped or specialised in any manner to suit different computation models. New structures are freely added in the interpretation part. This leads us to the detailed notations of components constituting music materials and interpretations.

5.3.1 Notations for musical materials

At the top level, musical material is a list of four lines (i.e. soprano, alto, tenor and bass). We explain our notations for the line below.

- Line: Line is an ordered list of events. There are two main components in the line a name and a list of events. A line constructor is given below.

line(Name, $[e_1, e_2, e_3, e_4, ...e_n]$**)**

Name:	A name of a line (e.g. soprano, alto, tenor, bass).
List of events:	This is a list of events $(e_1, e_2, ...e_n)$ in order of onset time. The requirement for order is just for convenience so that we don't need to sort the list every time we want to use it. Events is defined below.

- Pitch notations: Pitch notations are based on the conventional western music notations. We have decided to represent the pitch as a tuple:

⟨Pitch name, Pitch accidental, Pitch register⟩

Pitch name:	A class of $\{\hat{1}\ \hat{2}\ \hat{3}\ \hat{4}\ \hat{5}\ \hat{6}\ \hat{7}\}$ is used to represent pitch names.
Pitch accidental:	A class of $\{nat\ \sharp\ \flat\ \flat\flat\ \mathrm{x}\}$ is used to represent pitch accidentals.
Pitch register:	An integer number is used to represent registers of a pitch class. The group of pitches from the Middle $\hat{c}$ on piano up to the next $\hat{b}$ is referred to as register number 4. The upper set from $\hat{c}$ to $\hat{b}$ is the register number 5, and the one below middle $\hat{c}$ is the register number 3, and so on.
Example:	[1,nat,3] denotes the middle $\hat{c}$ in a piano.

- Note events: Note events are constructed from the pitch notation and notated time. We decided to represent the notated time as the duration and the onset time. Other pieces of information included at this level are those related to the notation (e.g. accent, staccato, trill, appogiatura, fermata, etc.) of each pitch event at the local level . A note event constructor is given below.

event(Pitch notation, Time notation, Feature slot)

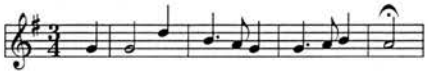
Pitch notation: The pitch notation contains information of pitch name, pitch accidental and register as defined above.

Time: The time notation is a list the onset time and the duration of the pitch (i.e. [Onset_time,Duration]). The onset time is the starting time of each pitch event and is calculated from the elapsed time since the start of the piece. In this implementation it is expressed using an integer; the value of 8 represents the duration of a crotchet in any tempo.

Feature slot: This slot has a list data structure. There could be as many concepts as needed for describing the note at the local level (e.g. fermata and other articulation marks). This local information is notated in the form of notation:Key:Value.

Example: event([1,nat,3],[0,8],[notation:anacrusis:8,notation:fermata:true])

At this point, we can give an example of the musical line constructed with all the components discussed above. The rendition of a melody line is given below.



```
line(soprano,[event([5,nat,4],[0,8],[notation:ksig:'G major',
                                     notation:tsig:34,
                                     notation:anacrusis:8]),
              event([5,nat,4],[8,16],[]),
              event([2,nat,5],[24,8],[]),
              event([7,nat,4],[32,12],[]),
              event([6,nat,4],[44,4],[]),
              event([5,nat,4],[48,8],[]),
              event([5,nat,4],[56,12],[]),
              event([6,nat,4],[68,4],[]),
              event([7,nat,4],[72,8],[]),
              event([6,nat,4],[80,16],[notation:fermata:true])])
```

The following notations are used in the feature slot above.

- `notation:ksig:G major` means key signature is in the key of G major.
- `notation:tsig:34` means time signature is in $\frac{3}{4}$ time.
- `notation:anacrusis:8` means anacrusis has duration of a crotchet value.
- `notation:fermata:true` means the note is notated with a fermata mark.

5.3.2 Notations for interpretations

At the top level, interpretations are grouped into list of objects (e.g. a chorale object, a phrase object). The term 'object' used in this implementation does not carry the same connotations as those used in the object oriented programming paradigm. Our objects are composed from slot of properties. There is no method implementation in the object. However, the absent of methods is not a requirement. The most useful property in our object implementation is that the super property and sub property in each object are updated automatically when an object is newly created or destroyed.

In each object, the interpretations are grouped under different general concepts as form, melody, harmony and texture as explained in 5.2.3. The groupings are arbitrary but must be consistent when adding new concepts to the interpretations. Different concepts could be assigned to the same musical materials to create multiple views. In this implementation, the interpretation is grouped in a chorale object and phrase objects.

$$\textbf{Interpretation} = \{Obj_1, Obj_2, ...Obj_n\}$$
$$\textbf{Obj} = \textbf{object(name, attributes)}$$

An object is constructed from a pair of name and its attributes. Attributes are notated as property lists. Properties are in the form of `Type:Name:Value`. `Type` can have any value of the following: form, melody, harmony or texture. Examples of attributes and their meaning are listed below.

- `form:id:chorale` means the object is identified as a chorale object.
- `form:super:root` means the object is the root.
- `form:sub:[phrase-1,phrase-2,phrase-3,phrase-4,phrase-5,phrase-6]` means the object has the following objects as its children.
- `form:sub:[]` means the object is at a leaf node ([] is an empty list).
- `form:info:'Aus meines Herzens Grunde No. 1'`: this attribute gives the name information of the chorale.
- `form:perform:[repeat(phrase-1,phrase-2),phrase-3,phrase-4,phrase-5, phrase-6]`: This chorale has a heard structure of phrase-1, phrase-2, phrase-1, phrase-2, phrase-3, phrase-4, phrase-5, phrase-6.
- `form:ksig:'G major'`: key signature is in the key of G major.
- `form:tsig:34`: time signature is in $\frac{3}{4}$ time.
- `form:first_strong_beat:anacrusis(8)`: anacrusis duration has a value of a crotchet note.

- `form:phrase_length:[168,240]`: the phrase has an interval span between the time of 168 to 240.
- `harmony:harmonic_rhythm:crotchet` means anacrusis has duration of a crotchet value.

5.3.3 Rendition of the input melody from figure 5.3

In this section, we render figure 5.3 into our score representation. The rendition here is our actual representation at the start of the harmonisation process. The system starts off the harmonisation process with a simple score structure given below.

5.3.3.1 Score representation

```
score([line(soprano,
           [event([5,nat,4],[0,8],[notation:ksig:'G major',notation:tsig:34,notation:anacrusis:8]),
            event([5,nat,4],[8,16],[]),
            event([2,nat,5],[24,8],[]),
            event([7,nat,4],[32,12],[]),
            event([6,nat,4],[44,4],[]),
            event([5,nat,4],[48,8],[]),
            event([5,nat,4],[56,12],[]),
            event([6,nat,4],[68,4],[]),
            event([7,nat,4],[72,8],[]),
            event([6,nat,4],[80,16],[notation:fermata:true]),
            event([7,nat,4],[96,8],[notation:ksig:'G major',notation:tsig:34,notation:anacrusis:8]),
            event([2,nat,5],[104,16],[]),
            event([1,nat,5],[120,8],[]),
            event([7,nat,4],[128,8],[]),
            event([6,nat,4],[136,16],[]),
            event([5,nat,4],[152,16],[notation:fermata:true]),
            ...
            ...
            event([7,nat,4],[464,8],[]),
            event([6,nat,4],[472,16],[]),
            event([5,nat,4],[488,16])],[notation:fermata:true]),
       line(alto,[event([blank,_A,_B],[0,504],[])]),
       line(tenor,[event([blank,_C,_D],[0,504],[])]),
       line(bass,[event([blank,_E,_F],[0,504],[])])],

      [object(chorale, [form:id:chorale,
                        form:super:root,
                        form:sub:[phrase-1,phrase-2,phrase-3,phrase-4,phrase-5,phrase-6],
                        form:info:'Aus meines Herzens Grunde No. 1',
                        form:perform:[repeat(phrase-1,phrase-2),phrase-3,phrase-4,phrase-5,phrase-6]]),
       object(phrase-1,[form:id:phrase-1,
```

```
                form:super:chorale,
                form:sub:[],
                form:tsig:34,
                form:ksig:'G major',
                form:phrase_length:[0,96],
                form:first_strong_beat:anacrusis(8),
                harmony:harmonic_rhythm:crotchet]),
object(phrase-2,[form:id:phrase-2,
                form:super:chorale,
                form:sub:[],
                form:tsig:34,
                ...
                ...
object(phrase-6,[form:id:phrase-6,
                form:super:chorale,
                form:sub:[],
                form:tsig:34,
                form:ksig:'G major',
                form:phrase_length:[432,504],
                form:first_strong_beat:anacrusis(8),
                harmony:harmonic_rhythm:crotchet])]).
```

5.3.3.2 Adequacy of score representation

The interpretation part of the score (see page 96) describes musical concepts of the music materials. The expressiveness of the interpretation is only limited only by the level of formal descriptions of the musical notation. This is very expressive as the limit is actually determined by the implementation language (i.e. logical language in our case) and the completeness of musical knowledge the system possesses. We argue that the logic representation scheme of the score has sufficient expressive adequacy and computation efficiency for the task.

5.4 Score Operations

We need to be able to manipulate knowledge contents in our score representation. In the musical material part, we need to be able to retrieve and edit musical materials and their interpretations. The information is retrieved and grouped into concepts of form, melody, harmony and texture. The system uses this information to derive more knowledge to solve the harmonisation problem.

We have defined our score constructor, line constructor and object constructor. The complete system will need destructors of these types plus other operations on

knowledge components in the score. We organise our discussion under three headings:

- Generic operations
- Manipulating musical lines
- Manipulating interpretation objects

5.4.1 Generic operations

There are some basic building blocks which are well developed in music theory (e.g. pitches, intervals, scales, chords). Hence, it is possible to treat them as generic concepts for many different musical genres. The operations on these basic building blocks are the same and are reusable for many genres in western classical music (e.g. chorales, string quartets, piano sonatas). We summarise these generic operations below:

5.4.1.1 Basic Operations

In the following basic operations, we operate on basic musical properties: pitch, time, duration and interval. We give a summary of their representation again below:

Pitch	: {1 2 3 4 5 6 7} × {nat ♯ ♭ 𝄪 𝄫 x} × Integer
Onset time	: Integers
Duration	: Integers
Interval	: Integers

- The basic operation of time: The time dimension is the landmark of note events. Basic arithmetic on the time is an essential requirement. Harris *et al.* [1991] define the following functions for time (add_{xy} and sub_{xy} refer to add and subtract operations, where x and y are each one of {t d} standing for time and duration respectively).

add_{dd}	: Duration × Duration → Duration
add_{td}	: Time × Duration → Time
sub_{tt}	: Time × Time → Duration
sub_{dd}	: Duration × Duration → Duration

- The basic operation of pitches: Pitch comparison is the basic operation of a pitch class. To compare the height between pitches (e.g. C4 is higher than C3). Basic comparisons are greater; less than; equal; greater or equal; less than or equal [Wiggins *et al.*, 1989].

eq $_{pp}$	: Pitch × Pitch → Boolean
gt $_{pp}$	: Pitch × Pitch → Boolean
sub $_{pp}$	: Pitch × Pitch → Interval
add $_{pp}$	: Pitch × Pitch → Pitch
sub $_{pi}$	: Pitch × Interval → Pitch
add $_{pi}$	: Pitch × Interval → Pitch

- Compute interval size: To calculate the distance between two pitches in terms of the number of semi tones apart [Wiggins *et al.*, 1989].

sub $_{pp}$	: Pitch × Pitch → Interval
sub $_{pi}$	: Pitch × Interval → Pitch
add $_{pi}$	: Pitch × Interval → Pitch

5.4.1.2 More complex Operations

More complex operations may be built from previously defined basic operations. This will involve new type of data structure, we summarise the types which appear in our examples below.

Interval Spelling	: { diminish augmented perfect major minor } × { unison second third fourth fifth sixth seventh }
Key	: { 'C major' 'A minor' 'F major' 'D minor' 'B♭ major' 'G minor' 'E♭ major' 'C minor' 'A♭ major' 'F minor' 'D♭ major' 'B♭ minor' 'G♭ major' 'Eb minor' 'F♯ major' 'D♯ minor' 'B major' 'G♯ minor' 'E major' 'C♯ minor' 'A major' 'F♯ minor' 'D major' 'B minor' 'G major' 'E minor' }
Functionality	: { tonic supertonic mediant subdominant dominant submediant leading }
Scale Degree	: { tonic supertonic mediant subdominant dominant submediant leading }
Chord	: { major minor diminish augmented seventh } × Root Pitch
Scale	: List of pitches
Metrical_structure	: Terms (see figure 5.2)
Figure_Bass	: Terms

The above examples are not the only way to represent musical structures in our interpretations. Many prolog terms will serve the same purpose. However, these are the representations adopted in our system. Our aim is to illustrate how the operations are implemented on top of these representations.

- Reasoning about more abstract properties of time: Higher abstract levels of musical time may be computed; for example, the structural significance of each beat in each bar (i.e. relative structural importance of the pitch with respect to the time). We give examples of these higher level operations on other properties of time below.

  getBeat/3 : Note_event × Line → Metrical_structure

- More operations on pitches: For example, a pitch D4 is a tonic in the key of D major or D minor, but it is a leading note in the key of Eb major. The predicate getScaleDegree/3 give the scale degree according to its context (e.g. the pitch is a leading note, a tonic, etc.). The other example is the sortPitch/2 predicate which sorts the pitches according to their height.

  getScaleDegree/3 : Key × Pitch → Scale Degree
  sortPitch/2 : List of Pitches → List of Sorted Pitches

- More operations on intervals: For example, an interval from C4–E♭4 is three semi-tones and it is a minor third interval. However, an interval of C4–D♯4 which is also three semi-tones should identified as an augmented second in this case. The predicate intervalSpelling/3 is use to identify the interval between two pitches within the appropriate context. The other example is the figureBass/3 predicate.

  intervalSpelling/3 : Pitch × Pitch → Interval Spelling
  figureBass/3 : List of Note_events → Figure_bass

- More operations on scales: For example, The predicate constructScale/2 is used to compute all the pitch members of any forty eight possible scale types which are twelve major scales, twelve natural minor, twelve harmonic minor and twelve melodic minor scales.

  constructScale/2 : Key → Scale

- More operations on transpositions: For example, the predicate pitchTranspose/3 or keyTranspose/3 allow the transposition of a key or a pitch up and down the required interval. The key transposition must maintain the correct pitch spelling in the new key (e.g. F♯4 in the key of C major should read C♯5 in the key of G major after transposition (not D♭5).

pitchTranspose/3	: Pitch × Interval → Pitch
pitchTranspose/3	: Pitch × Interval Spelling → Pitch
keyTranspose/3	: Key × Interval → Key
keyTranspose/3	: Key × Interval Spelling → Key

- More operations on chord: For example, the predicate constructChord/3 lists the pitches of the chord; the predicate getFunctionality/3 infers what functional context of a given chord in the given key is.

constructChord/3	: Key × Functionality → list of Pitches
getFunctionality/3	: Key × Chord → Functionality

5.4.2 Manipulating musical lines

We can retrieve data and write data into a line, or many lines. In the current implementation, a line is constructed from pitch events. The line is always associated with time. Therefore, line names, pitch events and time must be specified when new events are to be written to the lines. The line names and time must be specified when information is to be retrieved from them. The time could be specified as a time point or a time interval, depending on the nature of operations. We summarise three basic editing patterns as below. In the notations below read symbols $<, \leq, >, \geq$ as the usual numerical comparison operators, read $\text{Event}_{(T_s-T_e)1}$ as event 1 with start time at T_s and stop time at T_e.

- Retrieve information from one line or many lines at a time point: In this situation, an event or events which occur at that time point will be retrieved.

$$\textbf{Retrieved events}_{\ Ttp} = \{\textbf{Event}_{(T_s-T_e)1}, \textbf{Event}_{(T_s-T_e)2} ..... \textbf{Event}_{(T_s-T_e)n}\}$$
$$\text{for each event: } T_{tp} < T_e \text{ and } T_{tp} \geq T_s$$

- Retrieve information from one line or many lines at a time interval: In this situation, an event or events which occur at that time point will be retrieved.

$$\textbf{Retrieved events}_{\ (Tint1-Tint2)} = \{\textbf{Event}_{(T_s-T_e)1}, \textbf{Event}_{(T_s-T_e)2} ..... \textbf{Event}_{(T_s-T_e)n}\}$$
$$\text{for each event: if } T_s \geq T_{int1} \text{ then } T_s < T_{int2}$$
$$\text{if } T_s < T_{int1} \text{ then } T_e > T_{int1}$$

- Write information to one line or many lines at a time interval: To write something into a line or lines, it must be confirmed whether the line to be written is blank or the new data must replace the time span of the old data exactly.

The three basic manipulations of musical materials are retrieving, adding and deleting of musical information in the score.

getScoreLine/2	: Score → Lines
retrieveEventTpnt/3	: Line × Time point → Event
retrieveEventTint/3	: Line × Time interval → Events
writeEvents/3	: Line × Events → Line
deleteEvents/3	: Line × Events → Line

5.4.3 Manipulating interpretation objects

The interpretation is a property of the musical materials viewed from different perspectives, namely: form, melody, harmony, texture. Basically, a particular interpretation is a set of concepts describing music materials. The three basic manipulations of interpretation objects are retrieving, adding and deleting musical information in the score.

getScoreInterp/2	: Score → Interpretation
addInterpObject/3	: Interpretation × Object → Interpretation
retrieveInterpObject/3	: Interpretation × ObjectName → Object
deleteInterpObject/3	: Interpretation × Object → Interpretation

5.4.4 Behaviours of the operations

The generic operations tend to be reusable when the knowledge content is viewed from the same viewpoints. The combination of these operations is limitless. However the behaviour of a single operation or their compositions may be classified into three main categories:

- Operations which yield boolean true or false.
- Operations which yield measurements.
- Operations which yield a property output.

This idea is not new: Athanasiou [1995] proposes similar elements in his *PHOEVOS* system (i.e. three basic *Base-level*: rules, tests and heuristics). We argue that it is

sufficient to manipulate the knowledge content in a system with the three fundamental operation types above. We, therefore, construct our operations into three main groups:

- Rules
- Measures
- Tests

For example, we could construct an operation which gives an output of a plausible progression from the given pitch—an inference rule. Then there could be an operation which determines if the progression is a descending fifth—a test. The operation may not just give the response of true or false, but may give an output of the quality of the solution in a given criteria—a measure. We will discussed these three groups in detail in the next chapter.

5.5 Summary

In this chapter, we have presented our representation framework for representing musical structure knowledge. The metaphor of the framework is based on a concept of a conventional notations and their associated interpretations. Our *'Score'* represents the notations as the *musical material* part of the score, and represents the associated interpretations of the notations as the *interpretation* part of the score. The interpretation part could express multiple viewpoints and is hierarchical by nature.

The chapter closes with a discussion on the operations of the Score. The behaviours of the operations applied to the musical materials and interpretations are classified into three types: rules, tests and measures. These operations are fundamental in our knowledge based system. We discuss this in detail in the next chapter.

Chapter 6

Explicitly Structured Control

Introduction

In this chapter, we discuss the operations carried out on our musical representations and how they are built up into descriptions of *compositional processes* with *explicit control.* These operations involve reasoning about knowledge and introduce the issue of ways to organise the domain and control knowledge. In our system, the appropriate grain size of control is focused on the control of atomic rules, atomic tests and atomic measures.

This chapter provides full details of the system. This includes the system architecture and the control components of our control language. We organise the discussion under these main headings:

- System Architecture
- Control Primitives
- Meta Interpreter
- Library of Atomic Control Definitions
- Control Strategies.

6.1 System Architecture

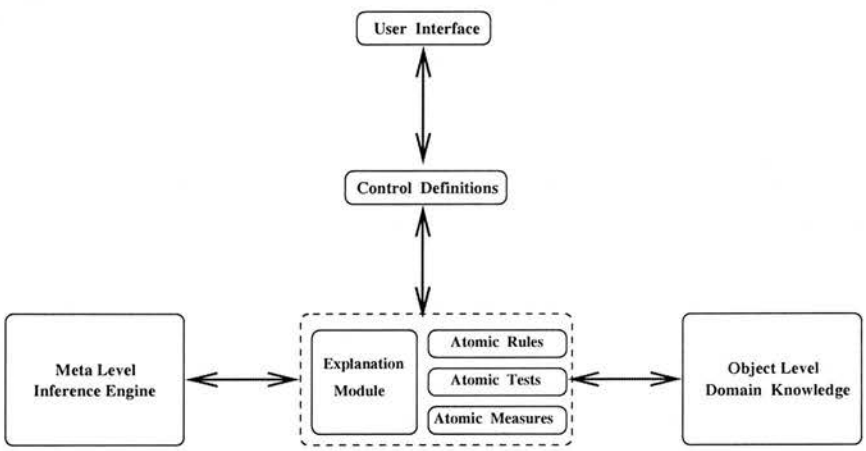
6.1.1 Terminology

The terms rules, tests and measures are introduced at the end of chapter 5 as three basic operations in the system (see page 111). We now give a concise meaning of new

terms we are going to introduce in this chapter.

- Control components: We classify our control components into two main classes (i) atomic control definitions and (ii) control primitives.
- Atomic control definitions: These are atomic rules, atomic tests and atomic measures. In our discussion, we use the terms:
  - atomic rules to describe rules at the meta-level (which will correspond to rules at the object level, the same idea is applied to tests and measures).
  - atomic control definitions as a general name to refer to the atomic rules, atomic tests and atomic measures in general.
- Control primitives: These are connectives which are specifically devised for our purpose. These connectives provide means to combine different atomic definitions or compound definitions together.
- Control definitions: A control definition can be constructed from atomic rules; atomic rules in combination with atomic tests or atomic measures or both using appropriate primitives. The control definition is a compound definition. Notice that we use the term a rule definition, a test definition and a measure definition to mean a compound definition constructed solely from atomic rules, atomic tests and from atomic measures respectively.

6.1.2 System architecture



Our representation framework is expressed in logic. We implement our control language using *Prolog* which also provides us with many extra features such as a depth first search

and a backtracking mechanism [Bratko, 1990; Clocksin and Mellish, 1981; Sterling and Shapiro, 1994]. Our system is constructed from four main modular components: (i) a meta-level inference engine; (ii) an object-level knowledge base; (iii) an intermediate layer which glues the object-level and the meta-level together and (iv) a graphic user interface.

1. The meta level inference engine: The meta level module is constructed with a domain independent principle. This fact might not be obvious initially since we only work with a single data type (i.e. the 'score') in this implementation. Discussion in page 197 shows how different data types can be employed. The meta-level inference engine examines control definitions which are our meta-level control language.
2. The object level domain knowledge: The object level consists of many modules specialising in different aspects of the domain knowledge. This allows a very flexible knowledge maintenance scheme.
3. The intermediate layer: This layer glues the object level and the meta level together. It provides a connection between the two levels. The atomic rules, atomic tests, atomic measures and the explanation module are fitted in this layer.
4. The graphical user interface: This provides a user friendly graphical interface for the harmonisation task. With this interface, user can select a new control definition or a new score input just by clicking, the harmonisation result can be displayed and printed out in a short score format at will.

6.1.2.1 Object-level knowledge and Meta-level knowledge

In our implementation, we look at our *score* (i.e. music materials and interpretations) as an object-level entity which musical domain knowledge is captured in one language (L_o). We define a meta-level language (L_m) which discusses the properties of the object-level in terms of *atomic rules, atomic tests, atomic measures* and *control primitives.* The context mapping between L_o and L_m happens in the layer of these atomic rules, atomic tests and atomic measures (see *intermediate layer* above). The object-level context–$Context_o$ is represented in the meta level as $Context_m$ (i.e. a connection is established between both sides). The $Context_o \in L_o$ represents object-level knowledge and $Context_m \in L_m$ represents meta-level knowledge. New meta theories for enhancing control could be added at this layer as well (adding meta knowledge in this area is part

of our proposed further work, see section 8.1.4: 'Reflection at the Meta-level').

6.1.2.2 Connections between levels

At the current implementation, the object-level and the meta-level are linked by a context mapping of ground terms between the two levels. For example, the meta context 'fill(passingNote)$_{meta}$' has its object-level mapping 'fill(passingNote)$_{object}$'. The causal connection by context mapping offers many advantages, e.g. abstraction power and hiding of low level control. However, expressiveness at the meta-level is directly dependent on the level of abstraction (i.e. the grain size of the meta-level language). The mapping does not have to be a one to one mapping, that is, it can be one to many or many to one (see more explanations in appendix D).

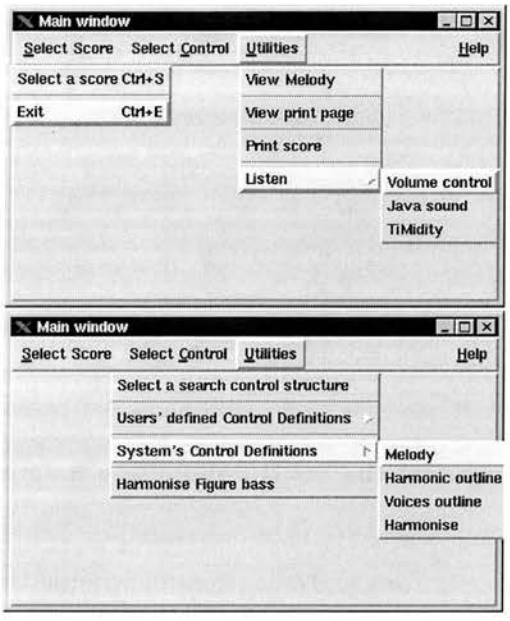
6.1.3 User Interface

Standard musical notation provides a powerful means for exchanging musical data. It is much easier to read music notated in a conventional score than to read a list of numbers and symbols representing the same musical data. In our system, harmonisation result can be displayed in conventional musical notations, the display can be invoked by including the 'display(harmonicOutline)' atomic rule in the control structure. A midi file of a harmonisation output can be created using the 'export2MIDI' atomic rule (see section 6.5.1 for more detailed description of these atomic rules). We also devised a simple user interface which allows interactions between human and machine with simple mouse-clicks.

6.1.3.1 The chorale harmonisation window

The chorale harmonisation window has four menus: Select scores, Select controls, Utilities and Help. The Select scores menu allows users to select a chorale melody from the score library. The Select controls menu allows users to select the predefined control definitions from the control library, and to harmonise a given melody with the selected control definitions. The Utilities menu allows us to view different output pages, to print the score to a printer and to listen to the chorale harmonisation output.

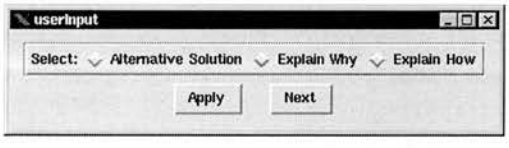
This interface is simple and menu driven. There should be no difficulty in exploring these windows. Explanation and information during run time appears in the blank space area of the window.



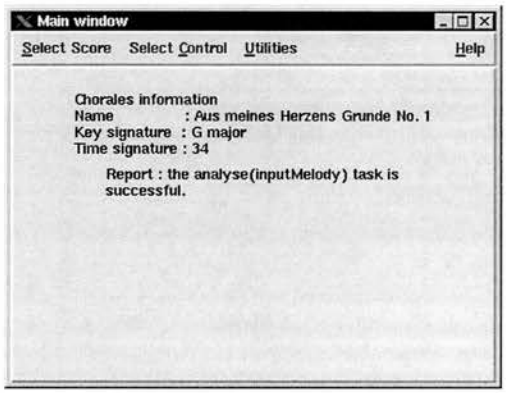
6.1.3.2 The interactive window

Our system also provides the **interactive** control primitive which will turn on the system interactive mode. The interactive mode aims to provide information on what has been done during the harmonisation process; to answer to *'Why'* and *'How'* questions from users. Currently, only selected control definitions are equipped with this interactive ability (due to time constraints–this aspect is tangential to the main topic of the research).

In an interactive mode, detail of the active process is constantly updated and users may ask for more explanation (i.e. with 'why' and 'how' questions).



Explanation is implemented as a separate extension to each atomic control definition. This allows a great flexibility in knowledge maintenance task. Our explanation module incorporates object level information to its explanation. The explanation generated from the same atomic definition varies according to the context of object level knowledge. We give snap shot examples of some explanation screen below.



6.1.4 Access to the score and the control definition libraries

Users compose new scores and new control structures and save them in the libraries. Currently, users can create a new score and a new control definition using a text editor. We provide templates for both the score and the control definition files below:

6.1.4.1 The Score library

We provide twenty five input scores from Bach's chorales. Users can experiment with various control structure using this material. New scores may be added into the library. However, the current implementation does not provide a GUI interface for this task. Users must create a file (*.scr) in the Prolog language. We have given the data structure of the input score in chapter 5.

```
:- module( score,[score/2] ).
:- Interp = [ object( chorale,Attribute0 ),
              object( phrase-1,Attribute1 ),
              ...
              object( phrase-N,AttributeN ) ],
   Lines  = [ line(soprano,SopranoElements),
              line(alto,AltoElements),
              line(tenor,TenorElements),
              line(bass,BassElements) ],
   SopranoElements = [S1,S2,S3,S4,S5,... Sn],
   AltoElements    = [BlankAlto],
   TenorElements   = [BlankTenor],
   BassElements    = [BlankBass],
   S1 = element( Slot1,Pitch1,Time1 ),
```

```
S2 = element( Slot2,Pitch2,Time2 ),
...
Sn = element( Slotn,Pitchn,Timen ),
Attribute0 = [ form:id:chorale,form:super:root,
               form:sub:[phrase-1,...,phrase-N],... ],
...
AttributeN = [ ... ],
assert( score( Interp,Lines ) ).
```

6.1.4.2 The Control library

We provide some examples of controls in the control library. These examples are constructed with four main control definitions: *viewMelody, viewHarmony, viewOutline* and *harmonise*. Actually, the first three control definitions are components of the *harmonise* control definition. Every control definition is constructed from the atomic control definitions given in the library (see section 6.5.

Users can experiment with various structures by creating their own control definitions. Our GUI interface provides up to five calls to users' defined control definitions (using keyword def1, def2,.. to def5), but of course, any number of definitions can be freely defined. Below is a structure of a control definition file (*.def).

```
:- module( control_def,[ definition/2 ] ).
   definition( def1,Control_block1 ).
   definition( def2,Control_block2 ).
   ...
   definition( defN,Control_blockN ).
```

6.2 Compositional processes

It is our interest to devise a way to express our thoughts in controlling the chorale harmonisation process. There seem to be two main approaches in carrying out the harmonisation task:

1. The problem may be solved in one main conceptual step. An analogy to this would be to imagine a painter painting a landscape by completing the four quarters of the painting one after another (which will be a very unusual approach). In this style, the final solution of a local area (each quarter) is completed in one conceptual step (before moving to the next local area and never come back for a

revision). Homogeneity of the whole picture is maintained by having constraints, preferences, or heuristics applied at the local level.

2. In contrast to the first approach, the problem may be solved in many conceptual steps. That is, the problem is examined at different abstract stages and the solution is obtained from refinement of the whole picture from abstract to literal. In this style, the homogeneity of the whole piece is not tied to the way global knowledge is coded rigorously at the local level. The homogeneity of the solution can be maintained at the appropriate stages (at the stage which is most natural to encode the knowledge).

Computationally speaking, neither of the above tactics is better than the other. However, the latter is very natural to humans. In constructing a knowledge based system, both tactics are observable. A system may focus on different tactics in its problem solving methods. However, users generally cannot change the control structure at this level. This is our main concern. In our opinion, flexibility in applying control is important. Users should be able to express the problem solving steps with flexibility.

In this thesis, *"compositional process"* is the term used to define activities we construct from atomic rules, atomic measures and atomic tests. The compositional process reflects the thoughts which are relevant to musical tasks. Naturally, hierarchical structure exists in the process, for example, harmonic implication is usually realised before actual voicing. In trying to model the compositional process, it is inevitable that the system must support the expression of hierarchical structure of tasks and flow of tasks. At the meta-level, our meta-interpreter deals explicitly with this requirement with control primitives such as **then**, **if—then_do**, **repeat**. The meta-interpreter also supports more complex control features such as the **branch–with**, **n_score–size**, **sort_score** primitives. This allows us to express control concepts such as heuristic prioritisation of partial solutions in concise and clear language. Now we discuss the three building blocks in our control mechanism.

- Control primitives: atomic control definitions are put together by means of control primitives.
- Meta-interpreter: atomic control definitions are linked to object-level with the meta-interpreter.
- Atomic control definitions: these are atomic rules, atomic measures and atomic tests.

6.3 Control Primitives

Logic provides only a limited set of connectives. In our application, it seems natural to use more connectives than *and, or, not.* What are the appropriate set of connectives which would allow us to combine the atomic rules, atomic tests and atomic measures for a composition process?

In our control language, we call these connectives 'control primitives'. In principle, the control primitives should allow the hierarchical structure and context dependency in knowledge to be captured and expressed. In this light, we define a set of *control primitives* which will allow us to express simple control behaviours by combining atomic rules, atomic tests and atomic measures together. Thus, the composition process may be described by this configuration. The choice of appropriate primitives is debatable. We argue that the following properties are the desired properties of the control primitives:

- Simplicity
- Non-redundancy: If a primitive can be built from other existing primitives, then the primitive is redundant.
- Completeness: Primitives should allow enough expressiveness to configure all necessary control definitions. The primitives should allow the heuristic dimensions described earlier (i.e. hierarchy and dependency) to be captured.

How should the hierarchical structure and the context dependency in knowledge be expressed? Generally, we connect two statements using primitives such as *then, or, and, if.* In the same manner, we devise a set of control primitives. We now summarise the control primitives implemented in our system, categorised by type of operation. More detailed explanations are given in section 6.4; examples of control structures constructed from these primitives can be seen in section 6.6.

- Unfolding of process: This describes how the definitions should be unfolded. The unfolding of one definition to many other definitions is used to describe hierarchical structure in musical knowledge. We further subdivide this class into:
  - Simple unfold: The primitives implemented here are **then**, **and**, **or**, **not** which are used to unfold atomic rules, atomic tests and atomic measures.

  - Conditional unfold: This describes a decision point for selecting different definitions based on some conditions. Context dependency heuristics can be expressed with primitives such as **if—then_do**; **if—then_do—else_do**.
  - Stochastic unfold: Non-deterministic behaviours may be introduced with the **branch** control primitive. This allows different branches to be selected with users bias.
  - Repetition: This allows the repetition of a definition. The **backtrackRepeat** and **repeat** describe the repeated activation of a definition.
  - Others: The predicate **first_score** only exploits the first output. It is useful for blocking chronological backtracking in a control block.

- Solution specifications: This allows us to specify what type of knowledge is to be output. The set of primitives are: **branch—with**; **filter—with**; **n_score—size**; **sort_score**. The specification of a solution is determined by *test definitions* and *measure definitions*. Test definitions may be viewed as hard constraints and the measure definitions may be viewed as soft constraints.
- Interaction: Some atomic control definitions are designed with interactive capability. The **interactive** primitive activates the interactive feature which is disabled by default. Once activated, interactive features in the atomic control definitions are invoked.

We believe the above control primitives allow us flexibility to explicitly structure useful control definitions. It is worth noting that the above primitives serve three principal purposes: allow the unfolding of control definitions (i.e. decomposition, condition, negation and repetition); allow the control over some detailed specifications of the solution (i.e. applying tests or measurements); and allow the control to be transferred between machines and users (i.e. interaction).

6.3.1 Syntax of control primitives

We argue that our control primitives are expressive enough for exploiting the hierarchical structure and the context dependency in knowledge. We give a full description of the syntax and semantics of these control primitives below.

<control_spec> ::= **definition**(<name>,<control_definition>)
<control_definition> ::= <atomic rule>
| <control_definition> **then** <control_definition>

	|	<control_definition> **or** <control_definition>
	|	**if** <test_definition> **then_do** <control_definition>
	|	**if** <test_definition> **then_do** <control_definition> **else_do** <control_definition>
	|	**first_score** <control_definition>
	|	**branch** [control(<control_definition>,<Pr>),...]
	|	**branch** [<control_definition>,...] **with** <measure_definition>
	|	**filter** <control_definition> **with** <test_definition>
	|	**n_score** <control_definition> **size** <number>
	|	**sort_score** <measure_definitions>
	|	**repeat** <control_definition>
	|	**backtrackRepeat** <control_definition> **until** <test_definition>
	|	**interactive** <control_definition>
<test_definition>	::=	<atomic test>
	|	<test_definition> **and** <test_definition>
	|	<test_definition> **or** <test_definition>
	|	**not** <test_definition>
<measure_definition>	::=	<atomic measure>
	|	<measure_definition> **and** <measure_definition>
<name>	::=	Prolog atoms
<number>	::=	integer
<Pr>	::=	real with probability values between 0.0 to 1.0

An *atomic definition* is a general term for the atomic rule, or atomic test, or atomic measure which is not decomposed further. When a control definition can be decomposed into many definitions, it is a *compound definition.* The compound definition has a simple structure of Head $\Leftarrow$ Body. The *Head* of each definition is the statement of the definition itself. The *Body* describes the detailed explicit control using other definitions and manifests the declarative statement of the head.

The atomic rule, atomic test and atomic measure definitions take the same input data type (i.e. the score) and return an output score, a Boolean value and a numerical measurement respectively. The score is passed among definitions in two fashions: sequencing and branching. When the control flows in sequences the output score of the

previous definition is the input score of the next definition and so on. At the branching point, each branch will take the same output score from the definition immediately before the branching point as its input knowledge.

6.4 Meta-Interpreter

A meta-interpreter treats programs as data. This provides an access to the reasoning of the programs' behaviour which can be very beneficial to the system. Prolog provides a powerful means for writing a meta program. Our meta interpreter is written in Prolog; the basic skeleton is based on the standard *vanilla* model [Sterling and Shapiro, 1994]. Our meta-level inference is classified as an amalgamated meta-level inference system according to van Harmelen [1989a]. In short, the system uses the same language in both the object-level and the meta level. The object-level context is mapped to a meta-level context (i.e. different representation languages).

The meta-interpreter used in our system is provided below. Its components are grouped into three main classes (i) solve (ii) solve_test and (iii) solve_measure.

6.4.1 Solve control definition meta interpreter

This is a top level solve predicate. A control definition is expanded to other control definitions if it is not an atomic control definition.

```
% solve( +ControlDef, ?Cinp, -Cout, ?Input, -Output )
% ControlDef : Control definition
% Cinp : Input control data
% Cout : Output control data
% Input : Input data
% Output : Output data
   solve( ControlDef,Cinp,Cout,Input,Output ) ←
            definition( ControlDef,ControlDefs ) ∧
            solve( ControlDefs,Cinp,Cout,Input,Output )
```

If the control definition is an atomic rule definition then the corresponding object level rule will be invoked (note that the rule in this interpreter is still in a meta-level language). The update_control records the successful applications of atomic definitions.

```
% solve( +rule:Rule, ?Cinp, -Cout, ?Input, -Output )
```

```
solve( rule:Rule,Cinp,Cout,Input,Output ) ←
        rule( Rule,Cinp,Cinp1,Input,Output ) ∧
        update_control( Rule,Cinp1,Cout )
```

6.4.1.1 'then'

The **then** primitive implies a sequence of operations of the control definition_1 is to be executed before the control definition_2. That is, if the control definition_1 succeeds, do the control definition_2.

```
% solve( +Control1 then +Control2, ?Cinp, -Cout, ?Input, -Output )
  solve( Control1 then Control2,Cinp,Cout,Input,Output ) ←
        solve( Control1,Cinp,Cinp2,Input,O2 ) ∧
        solve( Control2,Cinp2,Cout,O2,Output )
```

6.4.1.2 'or'

By default, the **or** primitive implies that the control definition_1 is tried before the control definition_2. This is an *or choice*. The control definition_2 will be tried only if the path under the control definition_1 fails to lead to a plausible solution.

```
% solve( +Control1 or +Control2, ?Cinp, -Cout, ?Input, -Output )
  solve( Control1 or Control2,Cinp,Cout,Input,Output ) ←
        solve( Control1,Cinp,Cout,Input,Output ) ∨
        solve( Control2,Cinp,Cout,Input,Output )
```

6.4.1.3 'first_score'

The **first_score** primitive allows only the first successful output knowledge to be explored. By default, all possible goals of each definition would be explored. The **first_score** primitive is implemented using a cut (!) in the Prolog language.

```
% solve( first_score( +Control ), ?Cinp, -Cout, ?Input, -Output )
  solve( first_score( Control ),Cinp,Cout,Input,Output ) ←
        solve_first_score( Control,Cinp,Cout,Input,Output )

  solve_first_score( Control,Cinp,Cout,Input,Output )←
        solve( Control,Cinp,Cout,Input,Output ) ∧ !
```

6.4.1.4 'repeat'

The **repeat** primitive allows the repetitive applications of a control definition. The **repeat** is iterative in nature. The structure **repeat** <control> only stops when the <control> fails.

```
% solve( repeat( +Control ),?Cinp,-Cout,?Input,-Output )
   solve( repeat( Control ),Cinp,Cout,Input,Output ) ←
          solve_repeat( Control,Cinp,Cout,Input,Output )

   solve_repeat( Control,Cinp,Cout,Input,Output ) ←
          solve( Control,Cinp,C1,Input,O1 ) ∧
          solve( repeat( Control ),C1,Cout,O1,Output )

   solve_repeat( Control,Cinp,Cinp,Input,Input ) ←
          ¬ solve( Control,Cinp,_,Input,_ )
```

6.4.1.5 'backtrackRepeat–until'

The **backtrackRepeat–until** primitive allows the repetitive applications of a control definition. The **backtrackRepeat** is iterative in nature and it stops when the test is true.

```
% solve( backtrackRepeat +Control until +Test, ?Cinp, -Cout, ?Input, -Output )
   solve( backtrackRepeat Control until Test, Cinp,Cout,Input,Output ) ←
          solve_Brepeat( Control,Test,Cinp,Cout,Input,Output )

   solve_Brepeat( _Control,Test,Cinp,C1,Output,Output ) ←
          solve_test( Test,Cinp,C1,Output )

   solve_Brepeat( Control,Test,Cinp,Cout,Input,Output ) ←
          ¬ solve_test( Test,Cinp,_,Input ) ∧
          solve( Control,Cinp,C1,Input,O1 ) ∧
          solve_Brepeat( Control,Test,C1,Cout,O1,Output )
```

6.4.1.6 'if–then_do', 'if–then_do–else_do' control primitives

This **if–then_do** primitive reads this way: if the test definition succeeds then carry out the control definition. If the control definition has a solution then it is successful. Otherwise it fails. The **if–then_do–else_do** describes conditional branching points. If

the test definition succeeds, then carry out the control definition_1. If the test definition fails then carry out the control definition_2.

```
% solve( if +Test then_do +Control, ?Cinp, -Cout, ?Input, -Output )
   solve( if Test then_do Control,Cinp,Cout,Input,Output ) ←
            solve_test( Test,Cinp,C1,Input ) ∧
            solve( Control,C1,Cout,Input,Output )

% solve( if +Test then_do +Control1 else_do _Control2, ?Cinp, -Cout, ?Input, -Output )
   solve( if Test then_do Control1 else_do _Control2,Cinp,Cout,Input,Output ) ←
            solve_test( Test,Cinp,C1,Input ) ∧
            solve( Control1,C1,Cout,Input,Output )

   solve( if Test then_do _Control1 else_do Control2,Cinp,Cout,Input,Output ) ←
            solve_not_test( Test,Cinp,C1,Input ) ∧
            solve( Control2,C1,Cout,Input,Output )
```

6.4.1.7 'branch'

The **branch** primitive allows users to bias the selection at branching points. This is done by tagging each control definitions with a desired probability. An example of an input +List with three control definitions tagged with probability of 0.1, 0.5 and 0.4 respectively is given below.

```
% List = [control(C1,0.1),control(C2,0.5),control(C3,0.4)]
% solve( branch +List,?Cinp,-Cout,?Input,-Output )
   solve( branch List,Cinp,Cout,Input,Output ) ←
            selectBranch( List,Control ) ∧
            solve( Control,Cinp,Cout,Input,Output )
```

6.4.1.8 'branch–with'

The **branch–with** primitive allows users to express their preferences at branching points. An input +List with three control definitions is given below. The +List here has a different structure to the one in the **branch** primitive.

```
% List = [ C1, C2, C3 ]
% solve( branch +List with +Measure,?Cinp,-Cout,?Input,-Output )
   solve( branch List with Measure,Cinp,Cout,Input,Output ) ←
            solveBranch( List,Cinp,Input,Output_list ) ∧
```

```
        sort_score( Output_list,Measure,Sort ) ∧
        memberchk( _Penalty:Output-Cout,Sort )
```

6.4.1.9 'filter–with'

The **filter–with** primitive allows us to specify the output knowledge with the test_definition. We can constrain our solution with test definitions.

```
% solve( filter +Control with +Test, ?Cinp, -Cout, ?Input, -Output )
   solve( filter Control with Test,Cinp,Cout,Input,Output ) ←
        solve( Control,Cinp,C1,Input,Output ) ∧
        solve_test( Test,C1,Cout,Output )
```

6.4.1.10 'n_score'

The **n_score–size** primitive allows a set of plausible solutions to be output and examined at the same time (the size of the set is determined by the size parameter). The predicate n_score/5 gives an Output_list with size equals Size, items in the list are alternative solutions of the ControlDef.

```
% solve( n_score +ControlDef size +Size, ?Cinp, -Cinp, ?Input, -Output_list )
   solve( n_score ControlDef size Size, Cinp,Cinp,Input,Output_list ) ←
        n_score( ControlDef,Size,Cinp,Input,Output_list )
```

6.4.1.11 'sort_score'

The **sort_score** primitive allows us to express heuristics to sort out the best solution from a finite set of plausible solutions. The solution are sorted according to preferences declared with measure definitions. The sorted output is ordered from the output with the smallest measurement (leftmost in the list) to the largest measurement (rightmost in the list).

```
% solve( sort_score +Measure, ?Cinp, -Cout, ?Output_list, -Output )
   solve( sort_score Measure, _Cinp,Cout,Output_list,Output ) ←
        sort_score( Output_list,Measure,Sorted ) ∧
        member( _Penalty:Output-Cout,Sorted )
```

6.4.1.12 'interactive'

The **interactive** primitive allows interaction between the user and the program. Control definitions which are built with interactive facilities will be activated with this primitive.

```
% solve( interactive( +ControlDef ), ?Cinp, -Cout, ?Input, -Output )
    solve( interactive( ControlDef ),Cinp,Cout,Input,Output ) ←
            toggle( interactive(true) ) ∧
            solve( ControlDef,Cinp,Cout,Input,Output ) ∧
            toggle( interactive(false) )
```

6.4.2 Solve_test meta interpreter

The corresponding object level test is invoked at this stage. The update_control records the successful applications of an atomic test definition.

```
% solve_test( +test:Test, ?Cinp, -Cout, ?Input )
    solve_test( test:Test,Cinp,Cout,Input ) ←
            test( Test,Cinp,Cinp1,Input ) ∧
            update_control( Test,Cinp1,Cout )

% solve_not_test( +Test, ?Cinp, -Cout, ?Input )
    solve_not_test( Test,Cinp,Cout,Input ) ←
            ¬ solve_test( Test,Cinp,_,Input ) ∧
            update_control( not Test,Cinp,Cout )
```

6.4.2.1 'or'

By default, the **or** primitive implies that the test definition_1 is tried before the test definition_2. The behaviour of this control primitive is the Boolean combination of a disjunction between the two tests.

```
% solve_test( +Test1 or +Test2, ?Cinp, -Cout, ?Input )
    solve_test( Test1 or Test2,Cinp,Cout,Input ) ←
            solve_test( Test1,Cinp,Cout,Input ) ∨
            solve_test( Test2,Cinp,Cout,Input )
```

6.4.2.2 'and'

The behaviour of this control primitive is the Boolean combination of a conjunction between the two tests.

```
% solve_test( +Test1 and +Test2, ?Cinp, -Cout, ?Input )
    solve_test( Test1 and Test2,Cinp,Cout,Input ) ←
            solve_test( Test1,Cinp,Cinp1,Input ) ∧
            solve_test( Test2,Cinp1,Cout,Input )
```

6.4.2.3 'not'

The **not** primitive implies that if the test definition returns true then the return value is false and vice versa.

```
% solve_test( not +Tests, ?Cinp, -Cout, ?Input )
    solve_test( not Tests,Cinp,Cout,Input ) ←
            solve_not_test( Tests,Cinp,Cout,Input )
```

6.4.3 Solve_measure meta interpreter

The corresponding object level measure is invoked at this stage. The update_control records the successful applications of an atomic measure definition.

```
% solve_measure( measure:Measure, ?Cinp, -Cout, ?Input, -Value )
    solve_measure( measure:Measure,Cinp,Cout,Input,Value ) ←
            measure( Measure,Cinp,Cinp1,Input,Value ) ∧
            update_control( Measure,Cinp1,Cout ) ∧
```

6.4.3.1 'and'

The **and** primitive allows the grouping of many measure definitions together. The final measurement is the sum of all measurements.

```
% solve_measure( +Measure1 and +Measure2, ?Cinp, -Cout, ?Input, -Measurement )
    solve_measure( Measure1 and Measure2,Cinp,Cout,Input,Measurement ) ←
            solve_measure( Measure1,Cinp,Cinp1,Input,Val1 ) ∧
            solve_measure( Measure2,Cinp1,Cout,Input,Val2 ) ∧
            Measurement = Val1 + Val2
```

6.5 Library of Atomic Control Definitions

The atomic rules, atomic tests and atomic measures are three basic atomic control definitions in our system. They are constructed from other lower level operations which operate at the object level.

- Atomic rules may be constructed from many lower level operations. An atomic rule takes an input score and outputs a new score. Plausible states in the search space are determined by the application of the rule.
- Atomic tests are aimed to provide a declarative statement of an interesting property which is either true or false. An atomic test provides a way to control the search (e.g. a test before a branching decision; a test for specific requirements after the application of the rules).
- Atomic measures may be constructed from many lower level operations. The measures aim to quantify the quality of solutions generated by the rules. An atomic measure returns a measurement of interesting properties according to a set criteria. The returned measurement provides a way to control the search (e.g. sorting, ordering or act as evaluation functions).

We need these atomic definitions to operate at a suitable grain size of the knowledge content. Lower level operations are encapsulated with lower level control and are hidden below the atomic rules, atomic tests and atomic measures. This abstracts away unnecessary detail and complexity at the grain size we are interested in. The control below each atomic control definition is embedded in the program structure and is not modifiable (unless the lower level code is rewritten). On the other hand, the control of the application of control definitions or atomic control definitions is not embedded at the object level and this allows the control to be modified at the meta-level (with great flexibility). The atomic control definitions have the following important properties.

- Encapsulation of input/output knowledge: The atomic rules, atomic tests and atomic measures are encapsulated with their input/output knowledge. Currently, our system achieves the encapsulation of input/output knowledge at this level with the uses of the score representation. This enables a reconfiguration of these components in any combination without the burden of matching and passing of input/output data between components.

- Non-volatile musical information: Score information generated by control definitions is added to the score. The information is non-volatile. However, the information is dependent on the backtracking mechanism in Prolog (i.e. it will be undone when backtracking).
- Connections between levels: The effects after the rule, test or measure is activated must be observable by the control. These connections allow the control to employ useful information in its appropriate meta control theories. To achieve this connection, the object-level context must be mapped to the appropriate meta-level context.

In our opinion, it is very convincing not to have the smallest grain size possible at these atomic control definitions. The fine grain size of knowledge content might enhance a microscopic examination but this would be at a higher cost of efficiency and it may not always be useful to go to a very minute detail. It is sensible to have the object-level and the meta-level architecture constructed just at the right grain size of the meta-level control theory. Therefore, the grain size of these atomic control definitions is the choice of developers. The grain size is, generally speaking, conditioned by the context of meta level representation the system developers aim to allow in their systems. The application of these atomic control definitions determines the way the search is traversed. All atomic definitions implemented in our system are listed below. They are grouped according to their relevant processes.

6.5.1 Atomic rules

We aim to have the atomic rules operate at the level which will abstract away unnecessary detail so that we can concentrate more on our search tactics (in terms of compositional process). The harmonisation of a chorale is achieved in four major steps:

1. Analyse an input melody
2. Outline each phrase with a harmonic plan
3. Sketch outline voices
4. Fill in the actual notes and other detail.

Currently, the first step is accomplished with a single atomic rule: *analyse(inputMelody)*. The second, third and fourth steps are carried out with a certain procedure which will control sub-processes in each phrase. A typical framework is as follows:

```
rule:selectPhrase(+Process,+Phrase)          ; select a phrase and process to work with
  rule:initialisePhrase(+Process)            ; initialise the phrase
    backtrackRepeat                          ; sub-process
    ( rule:outlineAlto(transition)
      then rule:outlineTenor(transition) )
    until
    ( test:property(altoLine,allFilled )
      and test:property(tenorLine,allFilled ) )
  closePhrase(+Process)                      ; close phrase
```

The sub-process must be enclosed inside an *initialisePhrase* and a *closePhrase* rules. In the above example, the alto line and tenor line are filled at the same time. It is also possible to complete an alto line for a whole phrase before filling in a tenor line. In this case, both sub-processes designed for filling in the alto line and the tenor line must be enclosed under separate *initialisePhrase—closePhrase* pair.

6.5.1.1 General facilities

There are two different output pages for displaying the harmonisation result. Options for *+DisplayPage* are given below:

display(choraleMelody)

Display an input melody of the chorales in a short score format.

display(harmonisationResult)

Display the harmonisation results, displayed outputs are harmonic plan notated in roman numerates; outline voices notated as a stemless note heads; or the final harmonisation result.

export2MIDI

Create a MIDI file (chorale.MID) of the current harmonisation, the file can be played using 'java sound' or 'timidity' commands on the user interface.

6.5.1.2 Outline harmonic progression

For convenience, we group the following atomic rules under the outline harmonic progression subsection. The idea of outlining all phrases with a plausible and coherent harmonic plan before attempting to fill in other detail is the way we choose to attack

the dependency problem of harmonic structure in a chorale. The control is therefore aimed at providing flexibility for the reasoning of harmonic plan in general.

selectPhrase(outlineHarmonicPlan)
selectPhrase(+Process,+Phrase)

Select a phrase for the 'outline harmonic plan' process. Without specifying a phrase name (e.g. *selectPhrase(outlineHarmonicPlan)*, the phrase is selected in order from the first to the last. Values of *+Phrase* are specified as phrase-Number (e.g. *phrase-1, phrase-2*, etc.).

analyse(inputMelody)

Examine the input score and do a preliminary preparation on the musical lines (e.g. break the melody line and group into phrases, mark the input melody with beat tags).

initialisePhrase(outlineHarmonicPlan)
closePhrase(outlineHarmonicPlan)

The initialise task sets up required parameters which are used in the outline harmonic plan process. The *closePhrase(+Process)* is a post-process knowledge maintenance procedure. The *initialisePhrase(+Process)* and the *closePhrase(+Process)* must be used together. All other sub-processes must be enclosed in these two rules.

outlineChord(intro)
outlineChord(body)
outlineChord(cadence)
outlineChord(cadenceA)

A harmonic plan of a phrase may be outlined using the *outlineChord(+Context)* rule. The body rule must be the last rule used in the combinations. We implement two rules for the outlining of a cadence task. The difference in them is that the outlineChord(cadence) tries the home key before other closely related key, while the outlineChord(cadenceA) tries the related keys before the home key.

outlineChord(transition)

The *outlineChord(transition)* breaks the grain size of the *outlineChord(body)* down. The *outlineChord(transition)* select a plausible transition chord from the leftmost uninspected soprano pitch. This rule may be used in place of the *outlineChord(body)* to gain more control during the design of the phrase's harmonic progression.

6.5.1.3 Outline four parts

selectPhrase(+Process,+Phrase)
selectPhrase(outlineVoices)
selectPhrase(outlineAlto); selectPhrase(outlineTenor); selectPhrase(outlineBass)

Select a phrase for the 'outlineVoices' process. The phrase is selected in order from the first phrase to the last phrase. Selecting 'outlineVoices' assumes that all voices in the selected phrase will be filled before working with other phrases. If one wants to partially work with some voices then moving to other phrases before coming back to complete other voices, one must specify a voice (e.g. 'outlineBass').

initialisePhrase(outlineVoices)
closePhrase(outlineVoices)

The initialise task sets up required parameters which are used in the outline voice process. The *close(+Process)* is a post-process knowledge maintenance procedure. The *initialise(+Process)* and the *close(+Process)* must be used together. All other sub-processes must be enclosed in these two rules.

outlineBass(intro); *outlineBass(cadence)*; *outlineBass(body)*

These three rules provide a compact means for outlining a complete bass line in a phrase. To gain more control at a finer grain size, users may use the outlineBass(transition) rules with appropriate tests and measures.

outlineInnerVoices

The inner voices are alto and tenor voices. This rule outline both alto and tenor voice at the same time starting from the beginning to the end of a phrase.

outlineAlto(transition)
outlineTenor(transition)

The transition implies a movement from left to right. These rule will outline the voice at the leftmost position in a phrase.

6.5.1.4 Fill in decorations

selectPhrase(fillinProcess)
selectPhrase(+Process,+Phrase)

Select a phrase for the 'fill in decorations' process. The phrase is selected from the first phrase to the last phrase fashion.

initialisePhrase(fillinProcess)
closePhrase(fillinProcess)

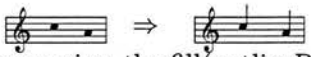
The initialise task sets up required parameters which are used in the fill in process. The *close(fillinProcess)* is a post-process knowledge maintenance procedure.

selectFillinPosition
deselectFillinPosition

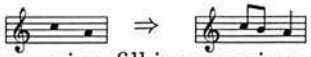
Select a position for filling in decorations. The position is selected from left to right starting on the first note of each phrase. The deselectPosition deselects the position selected for filling in decorations.

selectVoice(+Voice)
deselectVoice(+Voice)

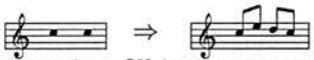
The *selectVoice(+Voice)* selects a voice: alto, tenor or bass. The *deselectVoice(+Voice)* deselects the selected voice.

fill(outlinePitch) 

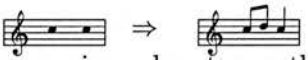
In a normal state of any voice, the fill(outlinePitch) fills in a crotchet with the outline pitch. The voice state is maintained at normal.

fill(passingNote) 

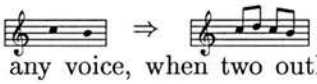
In a normal state of any voice, fill in a passing note between two outline pitches which are a third apart (in either ascending or descending direction). The voice state is maintained at normal.

fill(thirdDescPassing) 

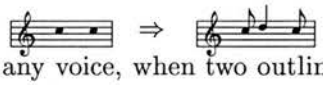
In a normal state of any voice, fill in two quavers (a third and a second above) between two outline pitches which are in unison. Change the voice state to the descending passing note state.

fill(neighborNote) 

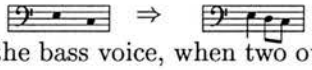
In a normal state of any voice, when two outline pitches form a unison, fill a neighbor note and maintain the normal voice state. The neighbor note can be either an upper or a lower neighbor note.

fill(neighborDescPassing) 

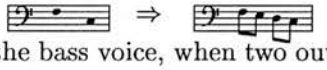
In a normal state of any voice, when two outline pitches form a descending second, fill an upper neighbor note and change the state to the descending passing note.

fill(neighborSuspension) 

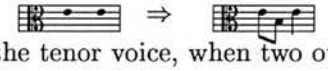
In a normal state of any voice, when two outline pitches form a unison, fill an upper neighbor note and change the voice state to suspension.

fill(descPassingI3Bass) 

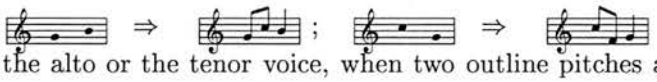
In a normal state of the bass voice, when two outline pitches form a descending third, fill an accented passing note at the second outline pitch. Change the voice state to the descending passing note.

fill(descPassingI4Bass) 

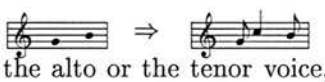
In a normal state of the bass voice, when two outline pitches form a descending fourth, fill two passing notes between the outline pitches. Change the voice state to the descending passing note.

fill(skipI4Alto) ; *fill(skipI4Tenor)* 

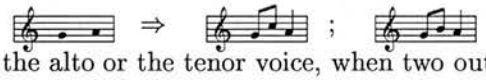
In a normal state of the alto or the tenor voice, when two outline pitches form a unison, fill a skip fourth down and maintain the voice state at normal. This pattern is only used to accompany a suspension on the voice above itself.

fill(escapeNote) 

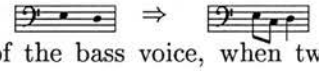
In a normal state of the alto or the tenor voice, when two outline pitches are more than a second apart, fill an escape note and maintain the voice state at normal. The escape note is filled above the second outline pitch for an ascending interval and below the second outline pitch for a descending interval.

fill(escapeSuspensionNote) 

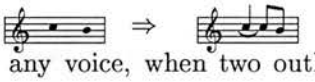
In a normal state of the alto or the tenor voice, when two outline pitches are more than a second apart in an ascending direction, fill an upper escape note above the second outline pitch and enter the suspension state.

fill(skipI3I4) 

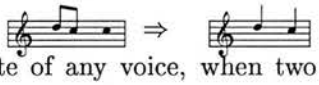
In a normal state of the alto or the tenor voice, when two outline pitches are a second apart in any direction, a third or fourth skip may be filled in the same direction. The voice state is maintained at normal.

fill(skipI3BassDown) 

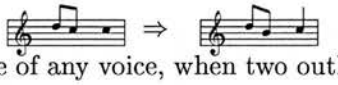
In a normal state of the bass voice, when two outline pitches are a second apart in any direction and the first bass pitch is not a root, then a third skip downwards may be filled in. The voice state is maintained at normal.

fill(suspension) 

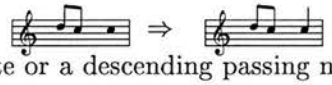
In a normal state of any voice, when two outline pitches form a descending second, the first outline pitch could be held till the next quaver. The voice state is changed to the suspension.

fill(holdSuspension) 

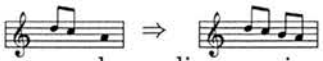
In a suspension state of any voice, when two outline pitches form a unison, the suspension pitch may be held for a crotchet duration. The voice state is changed to normal.

fill(decorateSuspension) 

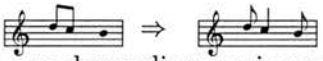
In a suspension state of any voice, when two outline pitches form a unison, the suspension may not resolve immediately but skip a third down and resolve in the next crotchet. The voice state is changed to normal.

fill(resolveStepDown) 

In a suspension state or a descending passing note state of any voice, the suspension may resolve a step down. The voice state is changed to normal.

fill(accentedPassingNote) 

In a suspension state or a descending passing note state of any voice, when two outline pitches form a descending third, the accented passing note may be filled in the second outline pitch. The voice state is changed to the descending passing note state.

fill(suspension) 

In a suspension state or a descending passing note state of any voice, when two outline pitches form a descending second, the first outline pitch could be held till the next quaver. The voice state is changed to suspension.

6.5.2 Atomic tests

We aim to have the atomic tests operate at a level which allows us to examine the property of the object level in the meta-level theory context. At the moment, we classify the test into two groups: constraint tests and property tests. The grouping is loosely based on the grain size of the tests. Generally speaking, tests which are local in nature would be grouped as property tests while tests which are global in nature would be grouped as constraints. We summarise these two groups in figure 6.1 and 6.2 respectively.

- constrain(+Arguments)
- property(+Concept,+Value)

6.5.2.1 Property tests

The property tests check the property of concepts. The figure 6.1 below summarises concepts and values of property tests currently available in the system.

property(phraseID, + Value)

This test check the value of the phraseID property of the current phrase. This allow the discussion about the phrase as the 'first phrase', the 'last phrase' as well as 'phrase-1', 'phrase-2', etc.

property(state(+Voice), + Value)

The state test checks the state of a voice. Each voice can be in a normal state; a suspension state or a descending passing note state.

Concepts	Attributes
phraseID	firstPhrase, lastPhrase
phraseID	phrase-N where N is 1,2,3...
state(alto)	suspension, descPassing, normal
state(tenor)	suspension, descPassing, normal
state(bass)	suspension, descPassing, normal
status(outlineChord)	allFilled
status(outlineAlto)	allFilled
status(outlineTenor)	allFilled
status(outlineBass)	allFilled
status(ornamentFillin)	allFilled
lowerBound(alto)	Pitch
lowerBound(tenor)	Pitch

Figure 6.1: Summary of test properties

property(status(+Process),+Value)

The status(+Process) test checks the status of a process. The value 'allFilled' means that all possible positions in the current phrase have already been filled.

property(lowerBound(+Voice),+Value)

This property checks if a voice is lower than a set value. At the moment, only the alto and tenor voice is equipped with this facility. The value is the pitch input in a meta level language.

6.5.2.2 Constraints

Constraints normally constrain more complex concepts; we think of constraints as a complex property test which involves a lot of object level tests. The advantage of working at a higher level abstraction from the atomic tests is that users do not need to define these concepts repeatedly. This is useful for concepts which are well defined and have always been complied with. The drawback is that explicit control cannot be applied to the level below this abstraction. In figure 6.2, we summarise constraints currently available in the system.

Arguments	Options	Arguments	Options
phraseHarmonicPlanOutline		skipLeadingNote	alto,tenor,bass
globalHarmonicPlanOutline		cadenceLeadingNote	alto,tenor,bass
melodicPatternXYXY	alto,tenor,bass	seventhPrepare	alto,tenor
repeatedHighCorners	alto,tenor,bass	oddSlotSeventh	alto,tenor,bass
tooManyRepeatedNotes	alto,tenor,bass	oddSlotUnison	
tritoneSpan	alto,tenor,bass	oddSlotCongestion	alto,tenor,bass
consecutiveSkips	bass	suspension	alto,tenor,bass
span7th9thSkips	alto,tenor,bass	oddSlotResolution	alto,tenor,bass
rhythmicPattern	alto,tenor,bass	consecutiveFifthOctave[1]	alto,tenor,bass
distantOctave[1]	alto,tenor,bass	parallelSecond[1]	alto,tenor,bass
oddSlotJump	alto,tenor,bass	exposedSecond[1]	alto,tenor,bass
susFromInessential	alto,tenor,bass	contrarySecond[1]	alto,tenor,bass
p4SkipDown	alto,tenor	exposedFifthOctave	bass
simultaneousSkip	alto,tenor,bass	directExposedOctave	bass
postSuspension	alto,tenor,bass	oddSlotDissonant[1]	alto,tenor,bass
doublingLeadingNote	alto,tenor,bass	xover[1]	alto,tenor,bass
rhythmicPattern	global		

[1]These constraints may be called without options.

Figure 6.2: Summary of constraints

Some constraints can be called with or without options. Options are attributes which serve to elaborate the constraints (e.g. the same constraint may be applied to different voices using different voice name options). It is possible to call all these constraints immediately after all voices have been filled (e.g. apply a call without an option at a harmony level). Generally speaking, we want to detect problems as soon as they occur. In such cases, applying constraints as soon as the voice is filled seems to be a good strategy. Users are free to configure the control as desired. However, the later the check is done, the further it is needed to backtrack to the problem stage.

constrain(phraseHarmonicPlanOutline)

The 'phraseHarmonicPlanOutline' constrains the harmonic progression of the chorales at a phrase level.

constrain(globalHarmonicPlanOutline)

The 'globalHarmonicPlanOutline' constrains the global harmonic progression of the chorales (e.g. the last phrase ends in home key).

constrain(melodicPatternXYXY,Options)

In any voice, the pitch pattern xyxy  is not allowed, unless it is enclosed in a sequence of wxyxyz in which both wxy and xyz move in the same direction (ascending or descending). In the case of the bass voice, the constraint is more strict. The progressions of both wxy and xyz must be a scale movement.

constrain(repeatedHighCorners,Options)

In any voice, it is desired to have only one highest point in the line.

constrain(tooManyRepeatedNotes,Options)

In any voice, the same pitch is allowed to be repeated three times in a row at most.

constrain(tritoneSpan,Options)

A tritone that spans over three notes is not allowed in the bass voice. Tritone spans over three notes in the alto or the tenor voice must be followed by a step in the same direction. A tritone that spans over four notes in all voices (except soprano) must be preceded or followed by a step in the same direction.

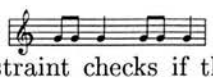
constrain(consecutiveSkips,bass)

Three consecutive skips in the bass voice are not allowed (unless the skip has an interval of a third)

constrain(span7th9thSkips,Options)

In any voice, a span of a seventh, a diminished or augmented octave, or any interval greater than or equal to a ninth spanned in three notes in the same direction are not allowed.

constrain(rhythmicPattern, Options)

The rhythmic pattern  is not allowed in any voice. The global rhythmic pattern constraint checks if the the pattern has emerged from the combination of SATB voices together.

constrain(distantOctave); *constrain(distantOctave, Options)*

Two octave intervals produced by any pair of voices are not allowed when they are separated by one or two quaver beats. This constraint extends the normal consecutive octave constraint further to rule out simple ornamentation tricks.

constrain(oddSlotJump, Options)

Odd slot is a term used in the CHORAL project (see [Ebcioglu, 1987, p 268]), Ebcioglu refers to a weaker quaver in a crotchet as the odd slot. This constraint states that: in a jump up-down of three quaver notes between two crotchet beats (start on the strong quaver), the downward jump should be within the interval of a third.

constrain(susFromInessential, Options)

If a suspension is prepared by an inessential note at a weak quaver beat, then the weak quaver must be a complete chord. There must be no other voice which sounds that preparation note and descends step down while the suspension is taking place.

constrain(p4SkipDown, Options)

A perfect fourth skip downward in a weak quaver between a unison alto/tenor is only used to accompany a suspension on that crotchet beat.

constrain(simultaneousSkip, Options)

If two voices skip simultaneously in the weaker quaver, it must constitute a parallel third and must individually move by a third.

constrain(doublingLeadingNote, Options)

No doubling of a leading note allowed.

constrain(postSuspension, Options)

After resolving from a dissonant suspension or descending accented passing note, the line may move upwards; or stay on the same pitch for another dissonant crash; or move downwards by step or third to the fifth of the next chord.

constrain(skipLeadingNote, Options)

If a leading note does not move to a tonic note (between a crotchet beat), then it is forbidden to decorate between the two notes with a skip quaver.

constrain(cadenceLeadingNote); *constrain(cadenceLeadingNote, Options)*

In a perfect cadence, if the leading pitch moves to the dominant pitch, then it should not be decorated with a passing note. In the $vii^o - I$ perfect cadence pattern, the bass should not be decorated with the root of the leading chord before the final tonic bass.

constrain(seventhPrepare, Options)

Do not decorate the preparation of a seventh.

constrain(oddSlotSeventh, Options)

A seventh may appear in the weak quaver of the crotchet beat, this seventh must be resolved down.

constrain(oddSlotUnison)

In this implementation, a unison is only allowed between the tenor and the bass. The unison cannot be approached in the weak quaver by step.

constrain(oddSlotCongestion, Options)

If three or more voices are filled in the quaver weaker beat, it should be a meaningful chord. Otherwise each filled-in must be a passing note. A meaningful chord is any chord allowed in that key (see chapter 4 for legal chords).

constrain(suspension, Options)

The resolution of the suspension cannot appear in other voices above the suspension voice. The resolution may appear in the voice below provided that

both voices are at least a ninth apart. The bass and tenor are exempt from this constraint.

constrain(oddSlotResolution, Options)

The resolution of a suspension or an accented descending passing note in the weak quaver must comply to the following: the chord formed in the weak quaver must be of the same chord; or a third down from that chord.

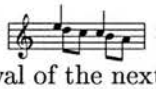
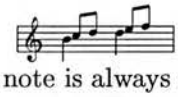
constrain(consecutiveFifthOctave); *constrain(consecutiveFifthOctave, Options)*

This constraint check for illegal consecutive fifth and octave. More information has been given in chapter 4.

constrain(parallelSecond); *constrain(parallelSecond, Options)*

Parallel seconds or compound seconds are forbidden. There is an exception to the compound seconds if it moves from a weak to a stronger quaver.

constrain(exposedSecond); *constrain(exposedSecond, Options)*

A second approached by parallel motion:  ;  must be prepared as illustrated. The second interval of the next note is always sounded in preparation. That is, if the voices are ascending the upper voice will sound the lower voice (next note) in preparation and if the voices are descending the lower voice will sound the upper voice (next note) in preparation.

constrain(contrarySecond); *constrain(contrarySecond, Options)*

A second approached by a contrary motion in a weak quaver is forbidden, unless it is formed an augmented second.

constrain(exposedFifthOctave, bass)

Exposed octaves or fifths formed by soprano and bass are not allowed unless one part moves by a step.

constrain(directExposedOctave, bass)

Exposed octaves or fifths formed by soprano and bass must not produce a harmonic interval progress from 7-8 or 9-8.

constrain(oddSlotDissonant); *constrain(oddSlotDissonant, Options)*

Exposed seconds or fourth or seventh arrived by skip in the weak quaver is forbidden, except that weak quaver forms a meaningful chord.

constrain(xover); *constrain(xover, Options)*

This constrains crossings between voices.

6.5.3 Atomic measures

The atomic measure returns measurement of properties. Some properties do not possess any default measurement. In this case, users must associate their own degree of measurements when calling the properties; measurement values are associated to the truth value of the properties. This idea can be extended to cover the property of the test definitions as well. By allowing users to use a test property as a measurement property, hard constraints in the form of test definitions can be turned into soft constraints at any level (determined by degree of measurement supplied by users). This is a very expressive technique.

6.5.3.1 Summary of Measures

measure(+Preferences)
measure(+Preferences,true(+Value))
measure(+Preferences,false(+Value))

We express preferences in our search by measuring the property of knowledge in terms of numerical measurements. In general, when preferences possess default measurements we use default measurements returned by properties when they are measured. When preferences do not possess default measurements, we must assign a value to the preferences. A call measure(+Preferences,true(+Value)) means if the property is true the measurement is +Value else the measurement is 0. The same idea applies to measure(+Preferences,false(+Value)). We summarise the properties currently available in figure 6.3.

property(preferredBassSuspension)

A suspension in the bass voice is desired only when it hides the second inversion on the suspension point.

Measurements	Associated values when false
property(preferredBassSuspension)	15
property(linearProgression(bass))	14
property(stepProgression(bass))	13
property(preferredRhythmicPattern(bass))	12
property(preferredUnisonOrnamentation)	11
property(preferredDissonantSuspension(tenor))	10
property(preferredDissonantSuspension(alto))	10
property(preferredChainedSuspension(tenor))	9
property(preferredChainedSuspension(alto))	9
property(linearProgression(tenor))	8
property(linearProgression(alto))	7
property(stepProgression(tenor))	6
property(stepProgression(alto))	5
property(undesiredTritoneSpan)	4
property(undesiredGlobalRhythmicPattern)	3
property(preferredTexture)	2
property(preferredParallelThirdSixth)	1
property(spacingInnerVoices)	
property(linearOutlineAlto)	
property(linearOutlineTenor)	
property(linearOutlineBass)	

Figure 6.3: Summary of measure properties

property(linearProgression(bass))

The bass line continues a linear progression (look back three notes from the current position).

property(stepProgression(bass))

The bass has moved by step.

property(preferredRhythmicPattern(bass))

The pattern quaver-quaver-crotchet is undesirable, if it starts on the strong beat of a bar.

property(preferredUnisonOrnamentation)

Unison in a weak quaver is undesirable.

property(preferredDissonantSuspension)

Dissonant second, fourth or seventh with the bass voice is desirable.

property(linearProgression(tenor))

The tenor line continues a linear progression (look back three notes from the current position).

property(linearProgression(alto))

The alto line continues a linear progression (look back three notes from the current position).

property(stepProgression(tenor))

The tenor has moved by step.

property(stepProgression(alto))

The alto has moved by step.

property(preferredTritoneSpan))

A tritone formed within 3 or 4 notes is undesirable

property(preferredGlobalRhythmicPattern)

A global rhythmic pattern quaver-quaver-crotchet is undesirable if it starts on a stronger beat of a bar.

property(preferredTexture)

In region spans for three crotchet, there should be quaver notes in some voices

property(preferredParallelThirdSixth)

Parallel third or sixth formed by two voices moving in step is desired.

property(spacingInnerVoices)
property(linearOutlineAlto)
property(linearOutlineTenor)
property(linearOutlineBass)

This set of property-test measures different properties of the outlined voices. The property 'spacingInnerVoices' measures different voicings in a chord. It gives more preference to a voicing with a wider spacing between lower voices. The property 'linearOutline' measures the progression of each voice. It gives a higher preference to the linear movement of a voice.

6.6 Control Strategies

In programs where the domain knowledge and the control knowledge are not explicitly represented, the control information is mixed with the procedural flow of the program. The main drawback of this approach is that the mixed control does not allow easy understanding of the control regime and does not easily allow control reconfiguration when a new control structure is required for a new process. This is because the control is implicitly embedded in the system. In our approach, the control (i.e. exploitation of control definitions at the meta-level language) and the domain (i.e. the application of rules, tests, measures and the score data at the object level) are explicitly represented. The control primitives allow us to explicitly express the configurations of the atomic rules, atomic measures and atomic tests. Reconfiguration is possible as a result from data encapsulation and modular structure between domain and control. This allows a great flexibility in expressing the control strategy over the search space.

Our search space consists of partial solutions, sub-optimal solutions and optimal solutions. The size of the search space is far too big for a complete search. As such, partial search is the only option. There are important questions to be considered: What part of the search space should we explore? What control knowledge is useful in guiding us to a good solution without going to the whole search space?

Search is effective when it is well guided. The way the search space is traversed is a crucial factor in determining the success of the task. There are two main areas when looking at how we can obtain the heuristics for guiding search. (i) The information available to us before the task is carried out and (ii) the information available while we are traversing the search space. The information in the first case is static and can

be determined beforehand. The information in the second case is dynamic and is only available at run time. The information in both cases may be grouped into three main classes:

- Data structure information: The information in this class discusses the data structure (in the meta-level language).
- Search structure information: The information in this class discusses the search activity (e.g. control applied, a search path).
- Solution state information: The information in this class discusses object level properties (in a meta-level language).

In our system, the explicitly structured control definitions defined by users are the act of applying meta level control to control atomic rules, atomic tests and atomic measures for the desired control behaviours. Currently, the reasoning at the meta level is under human control, that is, the configurations of different control definitions are predefined by users. The dynamic control is a very powerful control tactic, but its power is only apparent when the system has reflective capability. In this section, we concentrate on a meta-level control from a pre-defined control definitions. Only two of the three knowledge classes above are employed at this stage; (i) the search structure information and (ii) solution state information.

6.6.1 Control and Strategy

The exploitation of atomic control definitions guides the traversal of the search space at a defined area (i.e. defined by the control structure). We describe control strategy in a similar way to Meltzer [1970]. Meltzer describes the search with two components: the inference rules–I and the search strategy–Σ. We can express the compositional process in the following relationship:

$$\begin{aligned} &\text{Compositional process} = (I_R, I_T, I_M, \Sigma) \\ &\text{where } I_R, I_T, I_M : \text{Atomic rules, Atomic tests, Atomic measures} \\ &\qquad\qquad \Sigma \ : \text{Control strategy} \end{aligned}$$

The control strategy is obtained from the construction of atomic rules, atomic tests and atomic measures with appropriate control primitives. In a system, if the rule set is complete then we know that all the search space is addressable. However, if we are ready to accept a good solution which may not be the best, then there is no need to

look at the whole search space. This is a very practical practice in a problem of this scale. By configuring the search in certain patterns, different parts of the search space are explored. This technique reduces the size of the search space considerably. We describe the basic heuristics employed for constructing different control definitions as follows:

- Heuristics from hierarchical structures in knowledge: Hierarchical structure in knowledge is ubiquitous. Knowledge can be expressed in different grain sizes, from more general knowledge to more specific knowledge. The hierarchy of this arrangement is useful in our control strategies. By exploring the search space with general concepts before specific concepts, we get the shape of the solution in place before its details. The search space is narrowed down in such a way that fruitless parts are avoided early in the search process. This is an effective strategy for pruning the search space. For example, in our four-part writing model, the overall harmonic plan of the piece is at a higher hierarchy than the voicing in each harmony. Therefore, it is effective to explore the search space in terms of a harmonic plan before attempting to explore the voicing of each chord.
- Heuristics from contextual dependencies in knowledge: Musical knowledge may be closely interdependent. For example, if the previous two phrases are in the home key, then it is perhaps sensible to introduce variety by moving to another key in the current phrase even though it is possible to stay in the home key. The contextual dependency helps guide the search in the difficult situation when partial solutions are valid on a standalone basis, but they are inappropriate or wrong when they are put together. This dimension of knowledge is also ubiquitous in knowledge-based systems. Control strategy can be constructed from this dimension of knowledge for exploring dependent knowledge entities of the same dependent level at the same time.

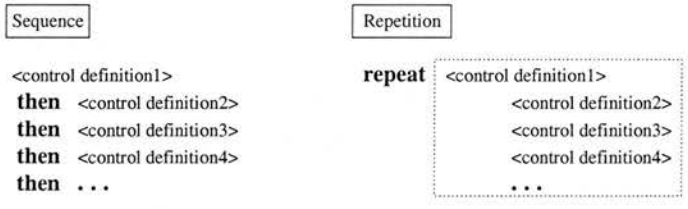
The hierarchy and dependency of the knowledge content are coexistent properties. The hierarchical structure breaks domain knowledge down into smaller units. The context dependency of these units is then visible. Search may employ these two basic heuristics to realise the desired search strategy.

We have discussed the design criteria of the control primitives based on the following classifications: unfolding of process; solution specification; and interactive control (see page 121). We argue that heuristics derived from hierarchical structure

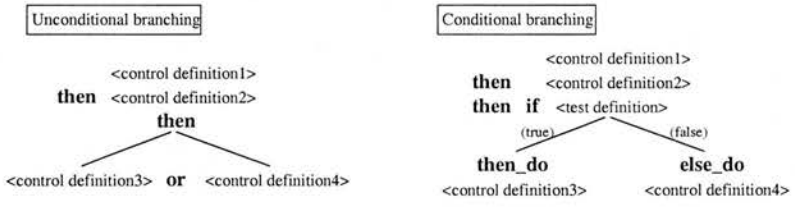
and dependency in knowledge are natural with these primitives. We discuss basic strategies constructed from primitives in these classes below.

6.6.1.1 Basic control: Sequences, Repetitions and Branches

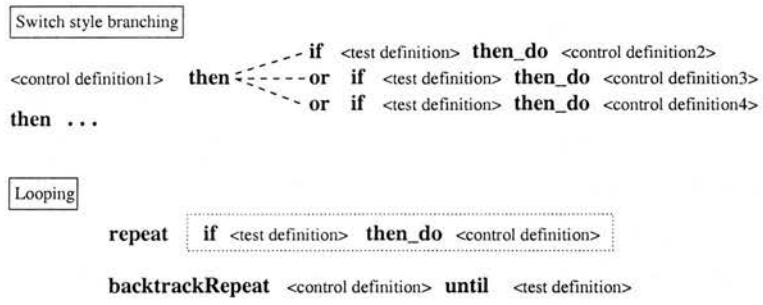
The common basic types of flow control are sequences, repetitions and branches. The following diagram shows a simple sequence of three control definitions and a repetition of these definitions.



We have allowed different types of branching to be expressed at the primitive level: unconditional branching and conditional branching are expressed using the **or**, **if–then_do** and **if–then_do–else_do** primitives.

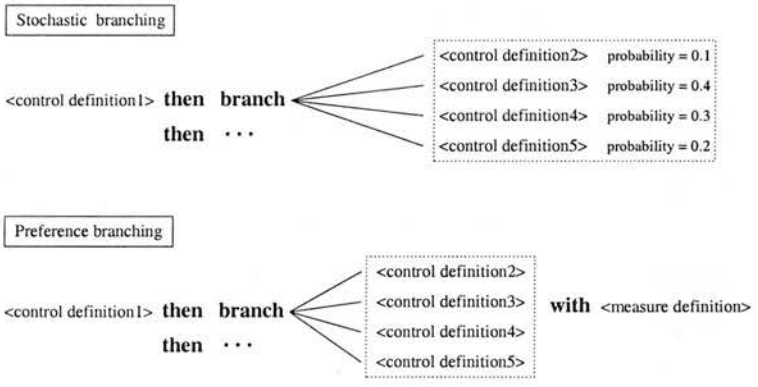


In their simple form, sequences allow us to express the ordering of process. Repetitions provide us with a means to repeat a certain process. Branches allow us to selectively direct the flow of control to different paths in the search space. Together, sequences, repetitions and branches provide us with a powerful means to control the flow of a process. More control structures may be constructed from these primitives. For example, we may construct a switch branching style by combining an **or** primitive with many **if–then_do** primitives; a **repeat** of an **if–then_do** control block would give a loop effect (e.g. a while loop, a for loop).



6.6.1.2 Basic control: Stochastic Branches and Preference Branches

Sometimes, we may want to select a branching point stochastically or with a preference. In this implementation, we provide two branching styles. They are **branch** and **branch–with** primitives. The **branch** primitive will select the branch to be further explored stochastically according to probability set by users. The **branch–with** primitive select the branch to be further explored with a preference set by users.



6.6.1.3 Basic control: Tests

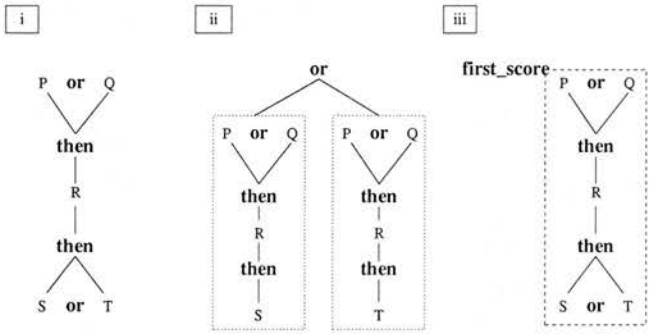
Tests provide us with means to examine if the knowledge content does or does not comply with our requirements. We can express the test before or after a control definition where the **if** <test_definition> **then_do** <control_definition> allows us to put the test before a control definition. Tests are used as constraints with the **filter–with** primitive. The **filter** <control_definition> **with** <test_definition> allows only specific solutions which comply to the constraints to be output.

6.6.1.4 Basic control: Measurements

Measures also provide us with means to examine the knowledge content. We may see tests as applying hard constraints and measures as applying soft constraints. Measures may be viewed as an evaluation function as well. Measurements could be employed to sort the output knowledge according to some preferences. The **n_score** <control_definition> **size** <number> and **sort_score** <measure_definition> does exactly this. Measures provide an effective means for a global control of solution quality by allowing us to keep track of different measurements in the course of the search.

6.6.1.5 Basic control: Backtracking and the 'first_score' primitive

Our system takes advantage of a backtracking mechanism in Prolog which is chronological. The **first_score** primitive may be used to skip a certain control block during the backtracking process. The structure of a control definition and a backtracking mechanism combined can produce different control behaviours. Consider the following control structures:



Assuming successful operations of all atomic steps, the behaviours of these control structures with backtracking mechanism are (i) P,R,S then P,R,T then Q,R,S then Q,R,T; (ii) P,R,S then Q,R,S then P,R,T then Q,R,T; (iii) either P,R,S then P,R,T or Q,R,S then Q,R,T (only the first solution of P or Q is considered).

The examples above are just some basic strategies constructed with available control primitives. Common search techniques such as hill-climbing search or best first search [Lugar and Stubblefield, 1989; Russell and Norvig, 1995] may be constructed using the proposed primitives. We believe the proposed model gives us an expressive way to explicitly construct a control at the meta-level. In the above example, some

control primitives are not discussed. However, their applications have already been explained earlier.

6.6.1.6 Basic strategy: Hierarchically narrowing the search space

To harmonise a given chorale melody, we may decompose the task into many major steps. We could write down the high level concept in our control language as follows:

definition*(harmonise,*
rule:analyse(inputMelody)
then *outlineHarmonicProgression*
then *outlineVoices*
then *fillinVoices).*

This is not the only way to approach the task. The point here is that the decomposition of the tasks above implies a strict ordering of tasks expressed with the **then** control primitive. The search space is hierarchically narrowed down with the decomposition used in this manner. At this level, the composition process can be conceived at a very abstract level. Each control definition may also be composed with other control definitions, or decomposed into many lower level definitions without worrying about lower level I/O data structure and lower level control for those definitions.

In this manner, hierarchical structure and dependency of compositional processes can be explicitly controlled. In the skeleton plan above, the *outline of harmonic progression* attacks the problem at a higher level in hierarchy than the *outlineVoices* and the *fillinVoices*. Each of these definitions may be broken down into different control structures and may hold more than one control structure at one time. We illustrate this in the harmonic plan outline process below:

definition*(outlineHarmonicProgression,*
repeat
(rule:selectPhrase(outlineHarmonicPlan)
then *outlinePhraseHarmonicPlan)).*

definition*(outlinePhraseHarmonicPlan,*
rule:initialisePhrase(outlineHarmonicPlan)
then *rule:outlineChord(cadence)*
then *rule:outlineChord(intro)*
then *rule:outlineChord(body)*
then *rule:closePhrase(outlineHarmonicPlan)).*

The *outlinePhraseHarmonicPlan* is a compound definition which may be defined in many different ways. We can compose the task so that it may do the introduction before the cadence or do the cadence before the introduction.

definition*(outlinePhraseHarmonicPlan,*
rule:initialisePhrase(outlineHarmonicPlan)
then *rule:outlineChord(intro)*
then *rule:outlineChord(cadence)*
then *rule:outlineChord(body)*
then *rule:closePhrase(outlineHarmonicPlan)).*

As the harmonic progression of a phrase can be constructed in various ways, a process is usually dependent on its preceding processes. Again, this is how the hierarchy and dependency come into play in the control.

6.6.1.7 Basic strategy: Maintain dependency in knowledge with Tests

We will use the same process above to show how dependency is handled. The list below shows some dependencies in the harmonic vocabulary which should be maintained in this style:

- Phrase should end with cadence pattern in some key (prefer perfect cadence).
- The end of the piece must be a perfect cadence in home key.
- The phrase before the last phrase should not end with tonic in home key.
- Harmony in the first phrase or first two phrase should establish the tonality in the home key.
- Chorales normally modulate to other key in the middle part of the piece.
- etc.

One way to maintain dependency in the knowledge is by constraints. The control primitive **filter** *Control_definition* **with** *Test_definition* allows us to express this. Suppose we have a test:constrain(harmonicPlanOutline) which performs the constraints above then we can write the above *outlineHarmonicProgression* definition in this way:

definition*(outlineHarmonicProgression,*
repeat
(rule:selectPhrase(outlineHarmonicPlan) **then**
filter *outlinePhraseHarmonicPlan*
with *test:constrain(harmonicPlanOutline))).*

By devising the tests in our control language, the tests can be plugged in/out as desired. This allows greater flexibility in the control structure.

6.6.1.8 Basic strategy: Applying heuristics with Measures

A heuristic is a best guess which does not guarantee an optimum solution. However, it is often necessary, especially in a large scale knowledge based system. A simple way to apply heuristics is by applying them using control structures (e.g. **then**, **or**). Heuristic preferences may be constructed with an or-branching control in which the branching is arranged so that the exploring of the more preferred branch is performed first. Preferences may be expressed using a stack based heuristic with backtracking facility. This tactic explores the most preferred dimension **then** the others but still allows a less preferred choice to be reached via backtracking. Other control structures (e.g. **if–then_do–else_do**) are also commonly used.

There are two major drawbacks associated with the above tactics. Firstly, the heuristic knowledge is mixed with the control. Secondly, the application of the heuristic is limited to only a local level (i.e. if not using tests to evaluate a global property). However, at a higher level in the hierarchy, the heuristics are quite clear and we happily express the heuristic with our control. To deal with detailed subtlety in our search space control, we devise measures. The measurement provides an effective means to keep track of the search space globally.

definition*(outlineBass*,
nScore *rule:outlineBass(transition)* **size** 20 **then**
sortScore *measure:property(linearProgression(bass)).*

Measurements are usually applied to a set of plausible solutions, where the set is sorted according to preferences before picking the desired solution (as in the example above).

6.7 Summary

In this chapter, we describe our atomic control definitions and control primitives currently implemented in our system in detail. In brief, the atomic rules are sets of operations which allow us to visit part of the search space by applying them. The atomic tests are sets of declarative properties of the domain which allow us to discuss their Boolean properties, while the atomic measures are sets of declarative properties of the domain which allow us to discuss preferences between them. The control primitives

are connectives that allow different types of atomic control definitions to be weaved together. Detailed evaluation of our control components will be discussed in chapter 7.

We have discussed how atomic control definitions and control primitives are formed into a process and how different areas in the search space are accessed with different configurations of control components. The control definition is constructed with explicit control which defines the skeleton of control structure at the meta level. Each so-constructed control definition is a theory expressed in meta language, the harmonisation task may be viewed as composing from (i) partial knowledge expressed in atomic rules(I_R), atomic tests(I_T) and atomic measures(I_M); and (ii) control knowledge(Σ) explicitly applied at the meta-level.

The search is controlled with heuristics derived from hierarchical structure and contextual dependency in the domain knowledge. At this stage of our work, the configurations of control components are defined by users. We summarise the important characteristics of our system below.

- We devise two classes of control components: atomic control definitions and control primitives.
- The composition process is manifested in the control definitions composed from control components.
- The compositional process is the expression of meta-level knowledge in terms of I_R, I_T, I_M and Σ.
- Input/output knowledge is encapsulated in atomic control definitions.
- The connection between the object-level and the meta-level is by means of context mapping of ground terms between levels.
- Control knowledge and domain knowledge are modularly structured.

Chapter 7

Discussion and Evaluations

Introduction

This chapter discusses and evaluates our work. The evaluation is carried out in two main areas: firstly, the harmonisation results; secondly, the control components in the system. We give a comparison between our results and Bach's original versions. More harmonisation examples from our system are included in appendix B[1].

7.1 Discussion

The experiment discussed in chapter 3 leads us to explore the issue of how the traversal of the search space can be controlled. The explicitly structured control is the result of this quest. The philosophy of the explicit control is simple: we want to control and guide the search at the level where we know how to control and guide it. Since different activities focus on different levels of control, lower level control may not be very interesting to exploit. These lower level activities are grouped under atomic control definitions. The explicitly structured control model controls and guides the search by structuring the atomic control definitions (i.e. atomic rules, atomic tests and atomic measures) into a desired control structure. The atomic rules exploit the search space under the explicit control frame given in the control structure. The atomic tests and atomic measures provide means to determine the property of contents in the solution

[1]Original harmonisation examples from the CHORAL system can be found in [Ebcioglu, 1987], original harmonisation examples from the HARMONET system can be obtained from http://i11www.ira.uke.de/~musik/Folien/. For convenience, we also included some examples from the CHORAL and the HARMONET systems in appendix B.

in a declarative manner. In this section, we discuss our system under the following headings:

- Areas where control knowledge can be applied
- Overhead problem in the meta-level architecture
- Choices of control components
- Interactions with the system
- Issues in search control.

7.1.1 Areas where control knowledge can be applied

At the meta-level, we suggest that there are four main areas where control information could be applied:

1. Control in inference engines: The inference engine could be enhanced with features dealing with specific meta-theories. For example, a simple vanilla meta interpreter may be enhanced with a depth bound control.
2. Control in control primitives: Each control primitive manifests control at the meta-level. For example, in the following **then** and **or** primitives:

$$\text{solve(A } \textbf{then} \text{ B)} \leftarrow \text{solve(A)} \textbf{ then } \text{solve(B)}$$

$$\begin{aligned} \text{solve(A } \textbf{or} \text{ B)} &\leftarrow \text{solve(A)} \\ \text{solve(A } \textbf{or} \text{ B)} &\leftarrow \text{solve(B)} \end{aligned}$$

   In this case, the **then** and **or** control primitives supply the control at the meta-level. They are applied to the control definitions (i.e. arguments A, B in the above examples). Each control primitive carries a specific connotation which is realised using the meta-interpreter.
3. Control in atomic control definitions: There are three types of control definitions in our system (i.e. atomic rules, atomic tests and atomic measures). Control knowledge may be implemented in these control definitions by (i) classifying the atomic definitions at the finer grain size to capture a better expressive control and/or (ii) loading atomic definitions with control information (i.e. data structure, search structure, solution structure).
4. Control in compound control definitions: This is a control built from combined atomic control definitions.

7.1.1.1 Control by means of a meta interpreter

This is the most natural place to think of control when talking about meta level control. There are two main properties when discussing the power of control of meta-interpreters: granularity and generality of the meta-interpreters.

- Granularity: For example, a meta-interpreter modelling unification is at a finer grain than a meta-interpreter modelling the selection of clauses [Sterling and Shapiro, 1994]. However, finer grain size may not always be a better choice and there is no fixed rule determining what is the right granularity. A trade-off between loss of efficiency and gain in control must be born in mind when deciding on the granularity of the meta-interpreter.
- Generality: The generality of the control applied at meta-interpreter could be meta theories of any type (see control strategies in 6.6). The domain specific control knowledge will improve efficiency and at the same time reduce the generality of the meta-interpreter. Again, there is no fixed rule determining what is the right level of meta-theories which should be applied here.

In our implementation, the meta-interpreter is at a very simple clause reduction level and is very general. We give our meta-interpreter on page 124. Meta-interpreters can be enhanced with more knowledge; examples of enhancement in meta-interpreter can be found in [Sterling and Shapiro, 1994; Bratko, 1990].

7.1.1.2 Control by means of control primitives

Control primitives may be implemented so that they carry implicit control knowledge in them. For example, we may implement a control primitive called **chooseBestRule** which will select the most appropriate rule (i.e. select a rule from the set of available rules, also assuming that the **chooseBestRule** can access useful information for selecting the right choice). Behaviour of the **chooseBestRule** is realised by the meta-interpreter. This is one possible area where we can load control knowledge into the system.

7.1.1.3 Control by means of atomic control definitions

In our system, we emphasise the application of control knowledge in the atomic definitions (at the meta-level). We see the application of control knowledge could be obtained

in two ways.

- Firstly, with a fine grain size in atomic control definition, the grain size of atomic control definitions determines the grain size of control we can explicitly compose in the control structure. Hence, what can be controlled is directly dependent on the grain size of the atomic control definitions.
- Secondly, with control knowledge loaded in each atomic definition at the meta level, the more informed the control definitions, the better their performance in object-level search (this is a result of the use of logic programming: the system can perform with partial knowledge and its performance increases when more information is available).

7.1.1.4 Control by means of compound definitions

The compound definitions carry control information in their structures. This dimension of control depends on how the control definitions are constructed. Its power is therefore closely dependent on the control components (i.e. primitives and atomic definitions) discussed earlier.

7.1.2 Overhead in the meta-level architecture

The efficiency benefits from the meta-level architecture is at the cost of efficiency loss from meta-level overhead. There is a certain point where increasing the meta-level effort will result in decreasing in overall efficiency [van Harmelen, 1989b]. However, we do not think it is a worthwhile effort to measure the exact overhead quantities in our system, if, indeed, this can be accurately measured at all, as the overhead at the object-level is dependent on knowledge organisation at the object-level and this is usually not well behaved. This is the main reason why we cannot pinpoint the desired properties of the meta-interpreter and other control components.

7.1.3 Choices of control components

Control primitives and atomic control definitions are the two possible areas where control knowledge could be applied to at the meta-level. As a matter of fact, these two areas are mutually dependent since the choice of one always influences the choice of the other to a certain degree. There is no fixed rule determining how to select the right

control components (i.e. control primitives and atomic control definitions). At the end, the decision on choices of control components depends on intuition and experience.

7.1.3.1 Choices of control primitives

As stated earlier (see page 121), the design of our control primitives are based on simplicity, non-redundancy and expressiveness of primitives. This is a general guideline that we use in this work. Why do we think simplicity is the best? We gave one example above for a plausible **chooseBestRule** primitive as an alternative choice. The **chooseBestRule** primitive is far cleverer than the primitives we have currently implemented in our system. The good point of a very clever primitive is that users are spared from constructing various lower control components at length to produce the same behaviour and the clever primitive is likely to reduce the meta-level overhead as well. However, if a slightly different behaviour is required, a new primitive is required and this approach (i.e. sophisticated primitives) could end up with more cost. In the end, we still believe that our design criterion is a good guide. Currently, the available control primitives appear to be sufficient for various control behaviours we can predict (see section 6.6.1).

7.1.3.2 Choices of atomic control definitions

We discuss the atomic control definitions in chapter 6. The control definitions currently implemented in our system operate on either a single *score* or a list of *scores*. However, our control primitives are not tied to any specific I/O and we could develop more control definitions and control primitives that operate on variety of data structures. We show the plausible enhancement and development of these features in our further work in chapter 8.

Variety and finer granularity in atomic control definitions promote detailed access to control information. This helps improving flexibility in control. We discuss control flexibility available from our present control components and point out some other control features which could be useful in the next section.

7.1.4 Interactions with the system

Currently, two types of input are required for the system, (i) the score input which contains input melody and other information (e.g. key signature, phrase structure) and

(ii) the control definition which describes the control structures of the search. Both of them can be prepared using a text editor; we have provided their structures in chapter 5.

We monitor the result after outlining the harmonic progression, outlining of voices and after filling in all details by means of the *display(choraleMelody)* and the *display(harmonisationResult)* control definitions. The interactions between the program and the users are very limited at the current stage. We have implemented the mechanism for explanation, but the full implementation is not yet realised at the current stage. The explanation of an internal state space during the search is very useful. Figure 7.1 shows an example when the system fails to outline inner voices and different harmonic progressions are tried out. It is quite hard to see why the task is not successful by looking at harmonic progressions and the outline basses. In this case, a better picture of the internal state will allow us to have more information of the search activities which could lead to a better control. Enhancement in this area is in-

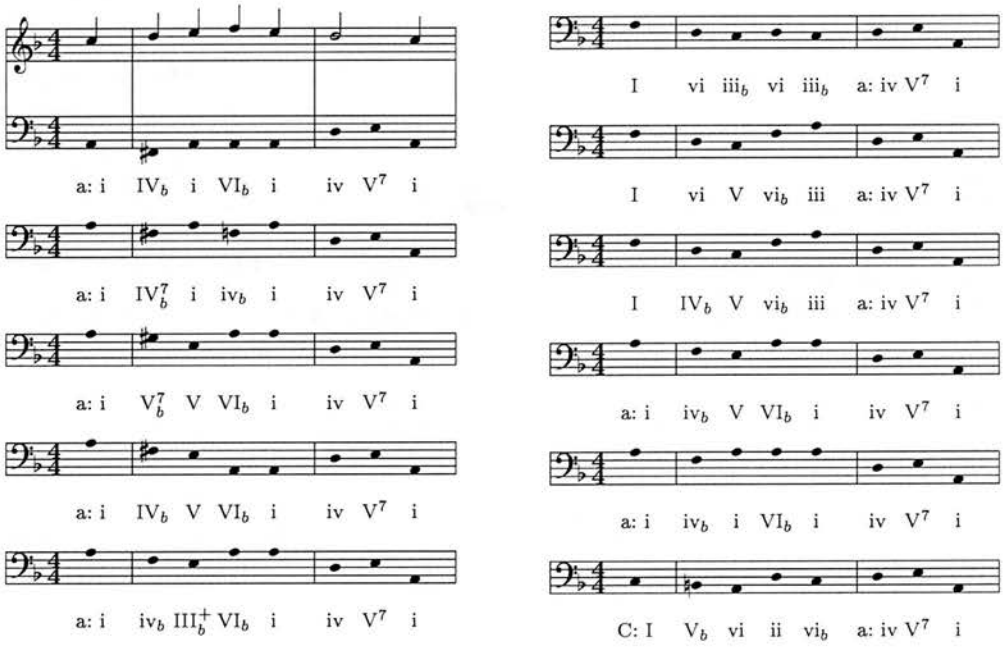


Figure 7.1: Lost in the search space

evitable especially when the control structure grows larger and becomes more complex. Although, generally speaking, the search space at the meta-level is smaller than the search space at the object-level and we greatly benefit by discussing the search at the

meta-level using atomic control definitions. As the system increases in size, the search space (at the meta-level) also increases. It is possible that the size of the search space at the meta-level itself could be huge as the system grows larger and we may need to resort to more control, and we need to have more information to guide search.

At the moment, only the **first_score** primitive provides a means to prune part of the search space away. The incorporation of meta theories such as search information, data structure information and more detail information about object-level behaviours could be very useful. These areas have not been rigorously investigated and we shall discuss some possibilities in our further work.

7.1.5 Issues in search control

Problem solving in AI can be viewed as search. It is clear that the size of the search space in our domain is far too large for a complete search. It is realistic to only partially search the search space. In our research, we look at the search control by emphasising the concepts of (i) hierarchical structure and dependency between knowledge constituents and (ii) locality and globality of control knowledge.

7.1.5.1 Hierarchy, Dependency, Locality and Globality

Our problem solving approach is based on a problem reduction technique. That is, we do not come up with a complete chorale harmonisation in one step. Instead, partial solutions are constructed and built up into a complete solution. This is a psychologically valid approach. However, this paradigm inherits some characteristics which play an important role in the search control. These characteristics (i.e. the concepts mentioned earlier) arise from the fact that a whole piece of knowledge holds extra implicit knowledge which is not discernible from smaller constituents. Local partial solutions may not see their globality dependency. Therefore, to reconstruct these smaller parts into a whole, there are some dependency rules which must be observed.

We stated earlier that $process = (I_R, I_T, I_M, \Sigma)$ where I_R, I_T, I_M correspond to atomic rules, atomic tests and atomic measures respectively; and Σ corresponds to the control strategy (see page 150). At any point in the search space, the I_R, I_T, I_M are bound to a local structure in their knowledge. This is dependent on the data structure employed in them. The Σ defines how concepts of locality, globality, hierarchy and dependency are incorporated in the search strategy. We list some options below.

- Organise global and local knowledge in the I_R, I_T, I_M: The I_R may incorporate tactics such as look-ahead, if the data structure employed permits this. For example, we may work out the harmonisation from left to right. At the point near the end of the phrase, we can look ahead and see if the cadence could be filled as a perfect cadence, and if it is appropriate to approach the cadence with a cadential $\frac{6}{4}$ elaboration. This is how the globality is included at a local level. Alternatively, the I_T may constrain this globality knowledge, or the I_M may have this globality knowledge.
- Expressing hierarchy and dependency in knowledge by means of structures in the control definition: We may explicitly structure the control to exploit the locality, globality, hierarchy and dependency in knowledge constituents. The order of processes reveals how they are hierarchically placed. It is a good strategy to put processes which are closely dependent close to each other to minimise backtracking. It is also a good strategy to construct the search in such a way that the search space is hierarchically narrowed down (i.e. shape of the solution before the final detail). We argue in the next section that this level of flexibility is provided in our control language.

7.1.5.2 Flexibility in expressing control

Generally speaking, flexibility in control may be viewed at different levels in the implementation. The flexibility between program and machine is valued differently from the flexibility between human and machine (via programs). In the first case, flexibility in control is machine architecture oriented. In other words, programming languages are designed to accommodate the hardware architecture. In the second case, flexibility in control is human thought oriented. This means that the emphasis is on the flexibility of the system to accommodate human thoughts in a natural way. Our system is orientated to human thought. We aim to have the system equipped with sufficient basic ingredients (i.e. atomic control definitions and control primitives) so that users can conveniently express their harmonisation thought without getting involved in control unnecessary to the musical thought. The main purpose of explicitly structured control is the bringing of these basic ingredients which are essential to the control up to the meta-level.

Flexibility in expressing control is apparent when knowledge is constructed into

rules, tests and measures. The control is expressed at the meta-level using control primitives which propagate control semantics in the composed structure. The flexibility of our control structure is the result of three factors:

- The nature of the control definitions: With a fine grain size of atomic rules, tests and measures, control can be applied at a detail level. A suitable grain size may not be the finest possible grain size in our application. Suitability to the reasoning context is what we are seeking. It is important that the abstraction should hide unnecessary control which is determined by the level of context associated with the domain. That is, keeping track of a counter is irrelevant to deciding which chord to use, and this type of control should be discouraged to appear at this level.
- The nature of the control primitives: Simplicity and expressiveness in control primitives are the desired properties. Compositional processes should be easily expressed using the primitives (i.e. unfolding, solution specification, interaction), see section 6.3.
- The I/O encapsulation mechanism of the control language: Flexibility of the control language is the desired property in our design. Our control primitives and our meta-interpreter are not tied to any I/O data type. It can be used with different I/O in a different domain.

7.1.6 Conclusion

The character of the overall control obtained from a system is the result of all the factors mentioned above. It should not be a surprise to find out that we have to incorporate these areas together for a practical purpose when implementing control knowledge.

Our main objective in seeking explicit control is to gain more flexibility in exerting control over how the problem solving methods are carried out. Are these implemented control components (i.e. control primitives and atomic control definitions) good choices for our objective? We think so, but we cannot claim they are the general choices for all purposes since we still find situations where we would like to be able to be more expressive in constructing our control. For example, using the current control components: (i) we cannot specify the exact harmonic plan since the actual decision on the harmonic plan is buried in the object level; (ii) we also cannot discuss many other details of object level properties. This is simply because we have only a limited set of

test definitions and measure definitions. At this moment, we feel our choices of control primitives suit our purpose well. However, we believe the choices of control definitions should be at a finer grain size (i.e. to yield more explicit control at the meta-level).

We would say that we expect improvement in all of these areas (i.e. the meta-interpreter, the control primitives and the atomic control definitions). However, it is very difficult to pinpoint the exact desired properties in each area. The discussions which follow serve to clarify why we think it is not possible to make a clear statement of the exact desired properties in each area.

7.2 Evaluations of Harmonised Chorale Outputs

In this section, we objectively evaluate the performance of our system. All chorale melodies in this evaluation are taken from *Bach 371 Harmonised Chorales* [Riemenschneider, 1941]. The aim of the evaluation is to explore the dimension of explicit control; what control definitions or control primitives could be added in to enhance flexibility and effectiveness of the system. Five harmonisation outputs from our system are included for this purpose. All harmonised chorale examples are produced from general control structures: we employ control structure #1 for harmonising chorales No. 1, 6, 26, control structure #2 for harmonising chorale No 217 and control structure #3 for harmonising chorale No. 80[2]. More harmonisation examples are included in appendix B.

7.2.1 Control structures used

Control definition #1 (see page 205) is designed as a general harmonisation procedure for most typical chorales. In other words, if there is no special interest in controlling particular features (e.g. texture, harmony), control definition #1 should give a plausible solution. We summarise the harmonisation process in control definition #1 below:

7.2.1.1 Analyse input melody

The definition `view_melody` below is composed of two rules: `analyse(inputMelody)` and `display(choraleMelody)`. The `analyse(inputMelody)` prepares appropriate data for further computation. The input melody is grouped into phrases. In each phrase, the melodic line is grouped as a sequence of sound events. Each sound event spans with

[2]control structures #1, #2 and #3 are included in appendix A.

the length of the desired harmonic rhythm (which is every crotchet in this case). The new data structure is added to the interpretation part of each phrase object.

```
definition( view_melody,
        rule:analyse(inputMelody)
        then rule:display(choraleMelody) ).
```

7.2.1.2 Select phrase

The control definition #1 select the phrase for the harmonisation task from the first phrase to the last phrase (i.e. using `repeat` primitive). The control definitions below are part of the control definition #1. The `harmonise` definition is composed of two definitions: the `view_melody` and the `fillinChoralStyle`.

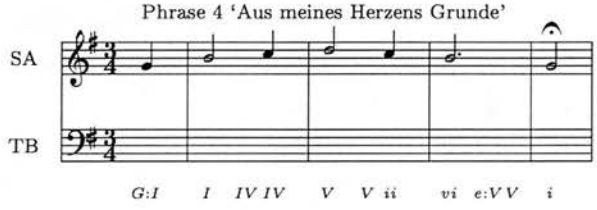
```
definition( harmonise,
          view_melody
          then fillinChoralStyle  ).

definition( fillinChoralStyle,
          repeat( rule:selectPhrase(fillinProcess)
                  then ( filter outline_phrase_harmonic_plan
                          with ( test:constrain( globalHarmonicPlanOutline )
                          and test:constrain( phraseHarmonicPlanOutline ) ) )
                  then outlineBass
                  then rule:display(harmonisationResult)
                  then outlineInnerVoices
                  then rule:display(harmonisationResult)
                  then rule:initializePhrase(fillinProcess)
                  then fillinView
                  then rule:closePhrase(fillinProcess)
                  then rule:display(harmonisationResult) )
              then rule:export2MIDI ).
```

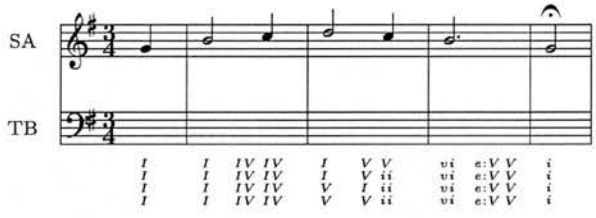
The `fillinChoralStyle` is a top level harmonisation strategy used in each phrase. The order of main activities are from `outline_phrase_harmonic_plan`, `outlineBass`, `outlineInnerVoices` and `fillinView`. In this presentation, we employ running examples from the fourth phrase of the chorale 'Aus meines Herzens Grunde'. Please refer to relevant details in the control definition #1 in appendix A.

7.2.1.3 Outline harmonic progression

The harmonic progression in each phrase is determined by working out a cadence, then chords at the beginning of the phrase and then the overall progression of the phrase (see definition `outline_phrase_harmonic_plan` of the control definition #1). The system does not attempt to put any note at this stage, just the harmonic progression. The harmonic plan in each phrase is determined locally at the phrase level. The overall harmonic plan of the piece is maintained using information such as position of phrase in the piece (see page 83) in the forms of tests and measures.



The bass line will be outlined next. If the bass line can not be completed with the current harmonic plan then the system will backtrack and try with alternative harmonic progressions. Belows are some progressions tried in this example.

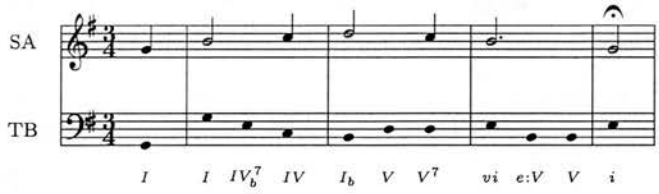


7.2.1.4 Outline voices

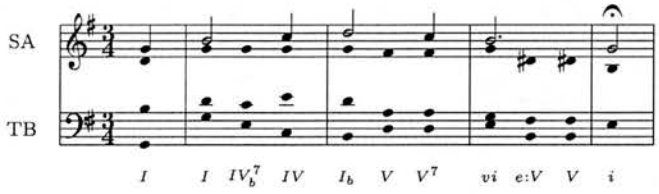
In each phrase, an outline bass and outline inner voices (i.e. alto, tenor) are filled. The control definition `outlineBass` works out the bass line starting from the cadence then the beginning and finally the body of the phrase. The control definition `outlineInnerVoices` fills inner voices from the start to the end of the phrase.

Filling in the bass line requires intuition as to what should the position of the first bass be in relation to the soprano line. At the object level, we incorporate information from the soprano line in selecting the bass. Two pieces of information from the soprano line are employed at the current implementation: the position of the lowest pitch in the soprano and the contour of the soprano line. In this example, the lowest pitch is $\hat{g}$. The system employs this information as a simple heuristic: the bass line should be at least an octave away from the melody line. The ascending contour at the beginning suggests

that the bass line should descend downwards and the descending contour at the cadence suggests that the bass line should ascend upwards. These are just heuristics which prioritise the generation of plausible solutions with respect to the above preferences. At this stage, the decision about chord inversions, doubling of pitches are made.

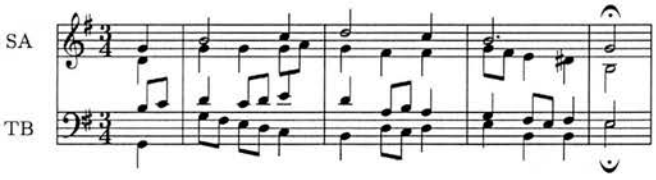


Once the bass line is in its place, filling in other voices are just a matter of filling the rest of the voices while observing the voice leading rules (see page 84).



7.2.1.5 Fill in final details

Finally, actual notes are filled in with decorations from the beginning to the end of the phrase. In the control definition #1, the fill-in sequences in each crotchet duration are from the bass, the tenor and the alto respectively (see `fillinView` definition). In filling detail decorations, the voice leading procedures are observed (see appendix C for details).



Each voice may be in any of these three states (i.e. normal, suspension and descending passing note). Each phrase starts with all voices in normal states. Depending on the melodic progression, different transitions (e.g. passing notes, neighbor notes, suspension notes) may be tried and the states of the voices may be changed to other appropriate states according to transition rules.

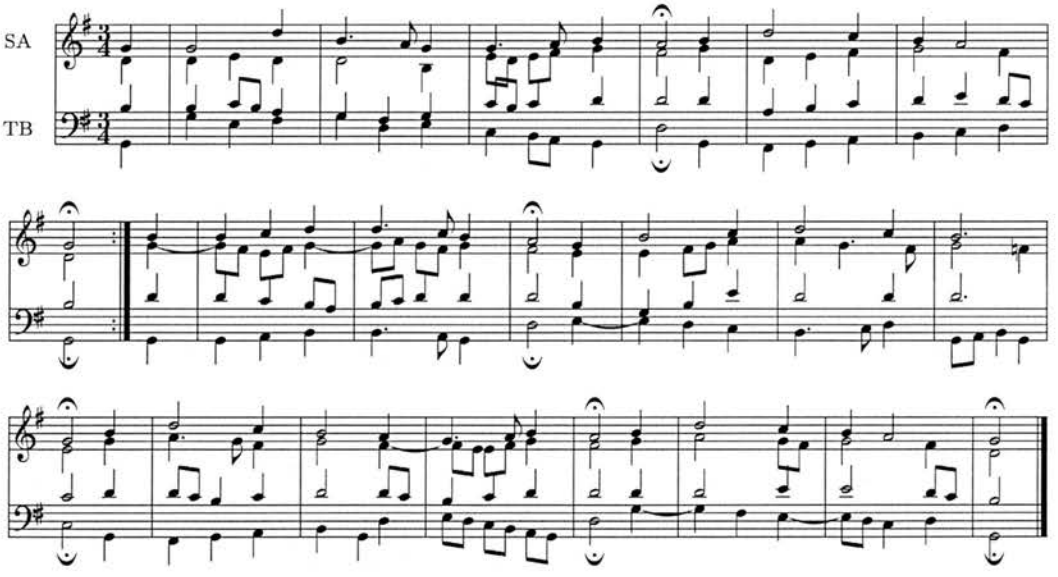
7.2.1.6 Other control structures

There is more than one way to construct a control structure for harmonising each chorale. Control definition #2 and #3 illustrate this point clearly. Control definition #2 has the same control structure as control definition #1, except that it fills in only crotchet notes without decorations; this gives a different texture to the output. We could incorporate stylistic details into our harmonisation in this way (provided that the control components also support the task). For example, we may want the bass part to be active with quaver movement while other voices move with a crotchet movement; or we may want to construct our control structure so that any two phrases with similar melodic shape produce different harmonisation outputs. In control definition #3, we introduce variation in the way decorations are filled in voices, we also employ a different rule in determining a cadence pattern (see more detail in appendix A). More harmonisation examples with different control structures will be discussed in section 7.3.1.

7.2.2 Harmonisation examples

7.2.2.1 Chorale: Aus meines Herzens Grunde

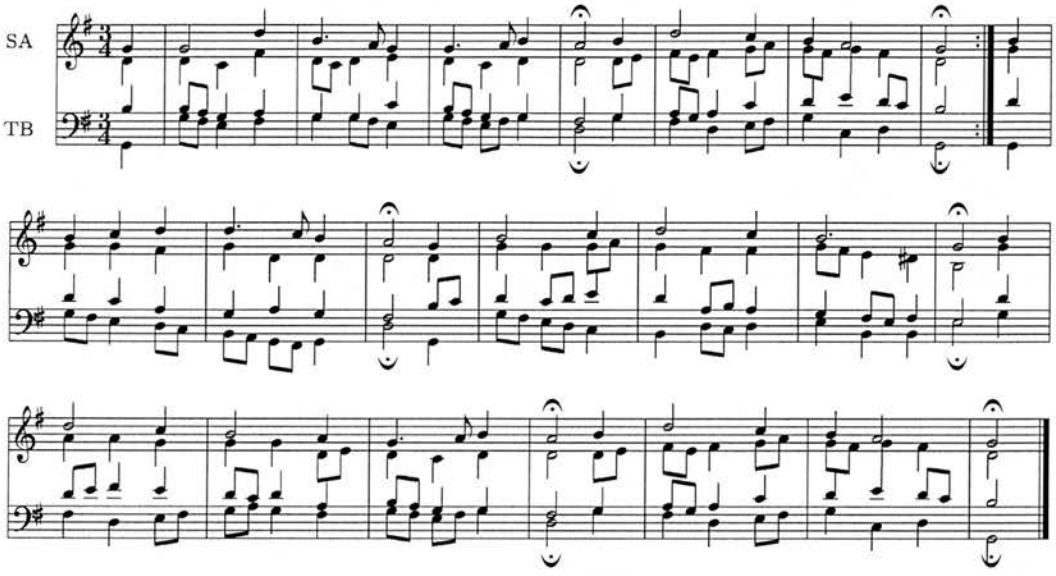
Aus meines Herzens Grunde No. 1 (Bach's original)



This chorale has six phrases with a repetition of the first two phrases (ABAB|CBAB).

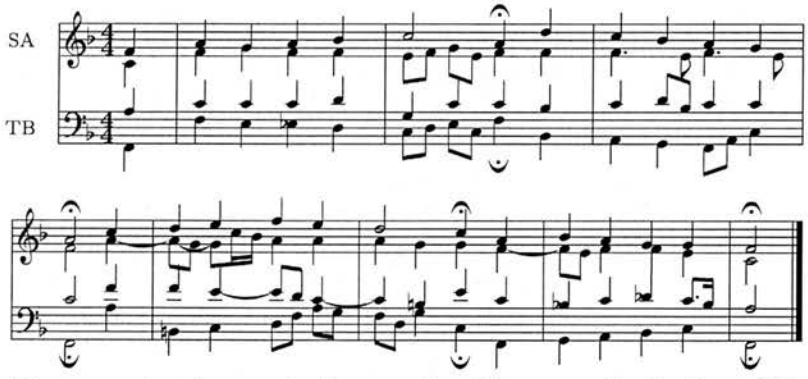
The original from Bach and the version from our system are given below for comparison. The control definition used in this harmonisation is included in appendix A.

Aus meines Herzens Grunde No. 1 (Control definition #1)



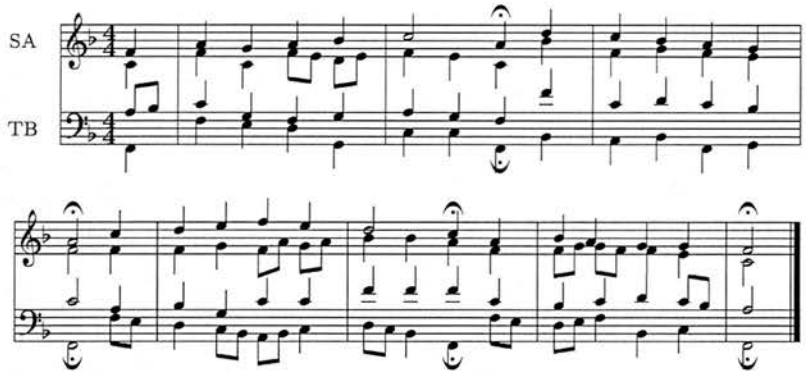
7.2.2.2 Chorale: Christus, der ist mein Leben

Christus, der ist mein Leben No. 6 (Bach's original)



There are two harmonisations under this name, in the key of F major (No. 6) and in the key of G major (No. 316), see [Riemenschneider, 1941]. However, they are quite distinct in terms of their melodies, time signatures and phrase structures. We choose chorale No. 6 for our example. This chorale has four phrases, the original from Bach and the version from our system are given below for comparison.

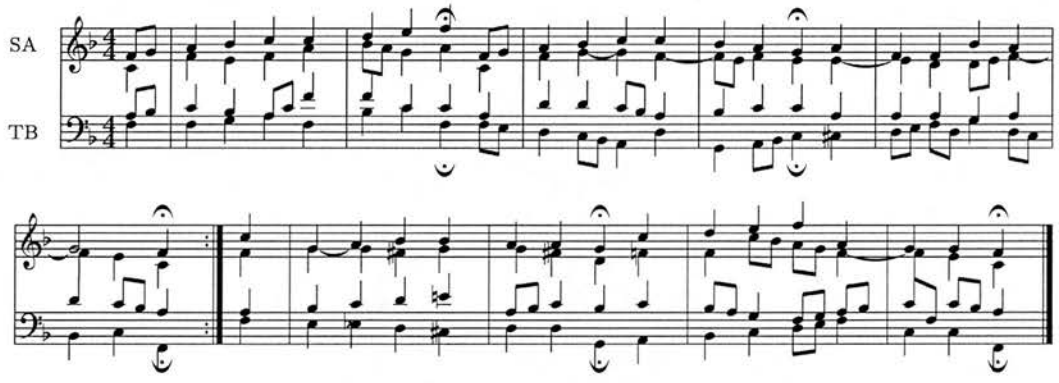
Christus, der ist mein Leben No. 6 (Control definition #1)



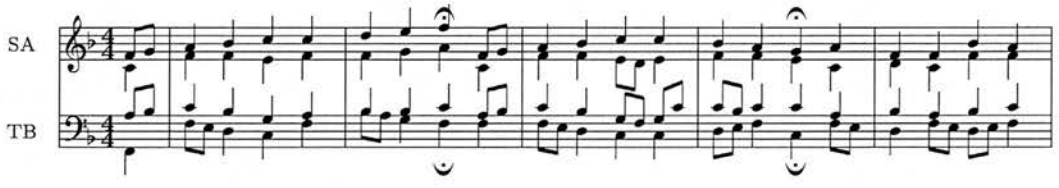
7.2.2.3 Chorale: O Ewigkeit, du Donnerwort

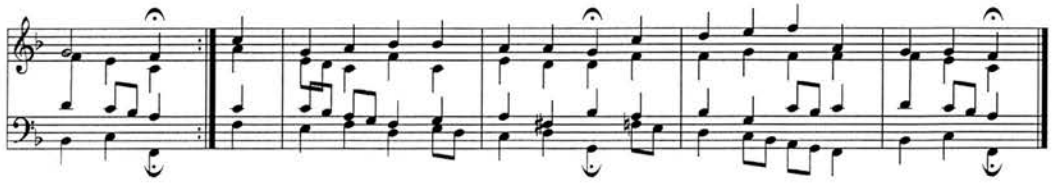
There are two harmonisations under this name, No. 26 and No. 274, see [Riemenschneider, 1941]. It has five phrases. The original from Bach (No. 26) and the version from our system are given below for comparison.

O Ewigkeit, du Donnerwort No. 26 (Bach's original)



O Ewigkeit, du Donnerwort No. 26 (Control definition #1)

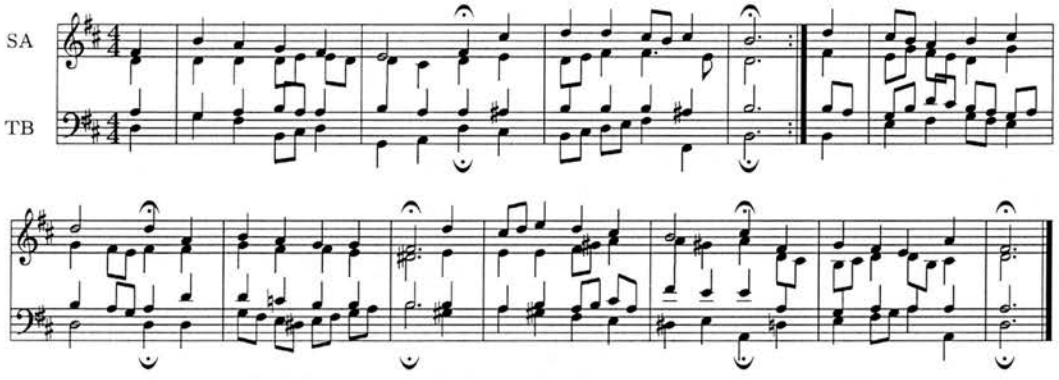




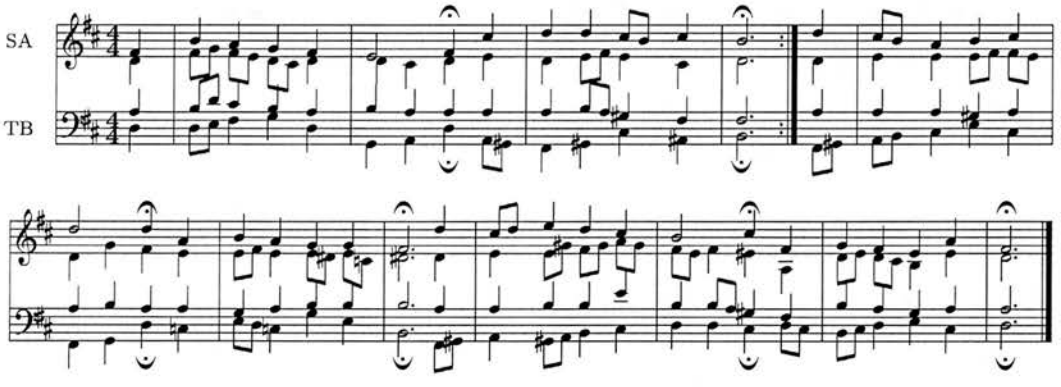
7.2.2.4 Chorale: O Haupt voll Blut und Wunden

Bach gave five harmonisations for this tune (No. 74, 80, 89, 98 and 345). This chorale has six phrases. The original from Bach (No. 80) and the version from our system are given below for comparison.

O Haupt voll Blut und Wunden No. 80 (Bach's original)



O Haupt voll Blut und Wunden No. 80 (Control definition #3)

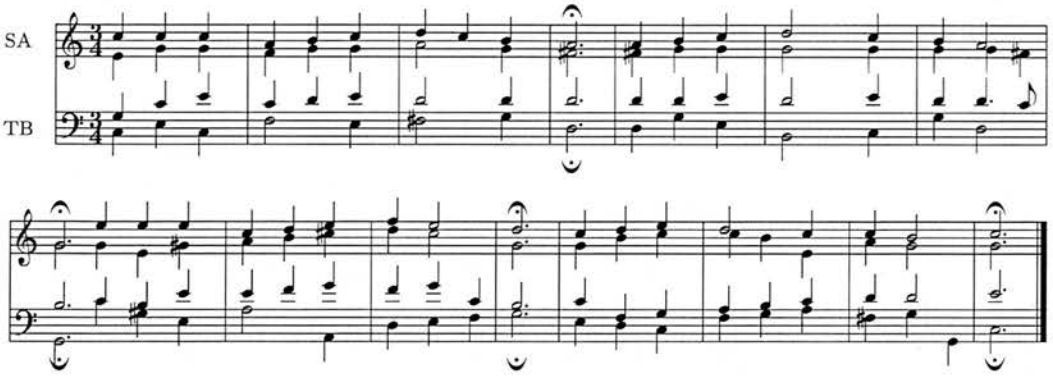


7.2.2.5 Chorale: Ach Gott, wie manches Herzeleid

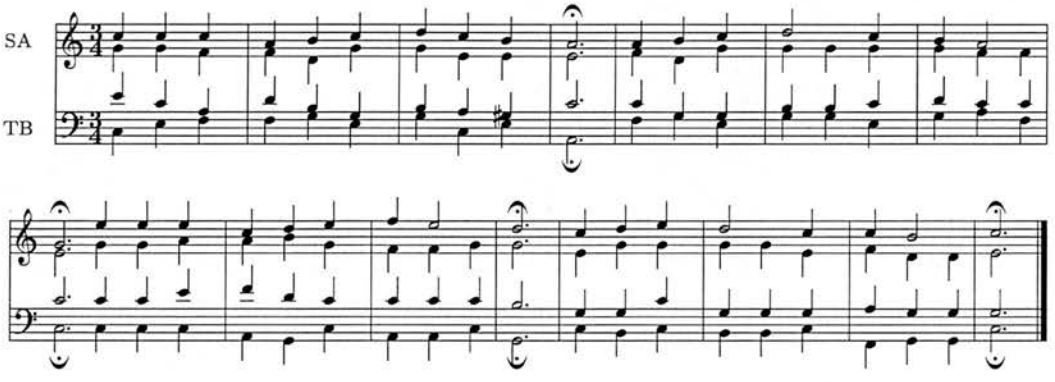
There are three harmonisations under this title (No. 156, 217 and 308). We choose No. 217 as our example to illustrate how the texture can be controlled with our control

structure. This chorale has four phrases. The original from Bach and the version from our system are given below for comparison. The control definition used in this harmonisation is included in appendix A.

Ach Gott, wie manches Herzeleid No. 217 (Bach's original)



Ach Gott, wie manches Herzeleid No. 217 (Control definition #2)



7.2.3 Evaluations

The five harmonisation examples given are the plausible harmonisation results produced by our system. We discuss the results from two points of view. Firstly we see the performance of our system with the implementor's eyes who is interested to see the effects of control structures applied upon harmonisation outputs. Secondly, we see this as how good the system performs in terms of their musical qualities. We seek comments from experts for our discussion regarding the latter aspect. All examples discussed here are included in appendix B.

7.2.3.1 System performance and the control

Although the control structure employed for these harmonisations are not specifically designed for each individual chorale. The overall result is reasonable. However, the following points are observed:

1. Generally, cadences in Bach's original harmonisation have a wide spacing between the bass and the tenor parts. The cadence in the fourth phrase of the chorales 'O Haupt voll Blut und Wunden' (Bach's version) is the exception here. Generally, a wider spacing in the lower voices is a general practice. In our first example (chorale 'Aus meines Herzens Grunde'), the cadences of the first, third and fifth phrase have closed spacings between the tenor and the bass parts.
2. Generally, in Bach chorales, no two phrases are harmonised with the same harmonic structure and the same voice. Our first harmonisation example (chorale 'Aus meines Herzens Grunde') shows a partial repetition between phrase one and phrase five, and an exact repetition between phrase two and the last phrase.
3. There is more than one way to write a cadence for a given phrase. The decision is dependent on many factors (e.g. What phrase is it? What is the cadence of the previous phrase? What type of cadence is it?). For example, we know that Bach seldom used plagal cadences in his chorales (see page 71). It may be more appropriate that the third phrase of chorale 'Christus, der ist mein Leben' should be in a perfect cadence in C major instead of a plagal cadence in F major.
4. As in the previous observation, from time to time, we see the system harmonises the chorale with a reasonable choice but not a better one. For example, the last four bass notes from the second phrase of chorale 'Christus, der ist mein Leben' are $\hat{\text{b}}\flat$, $\hat{\text{f}}$, $\hat{\text{g}}$, $\hat{\text{f}}$. The appearance of $\hat{\text{f}}$ crotchet before the final $\hat{\text{f}}$ reduces the effect of the cadence. The system does not choose to put $\hat{\text{a}}$ since this will lead to parallel octave of $\hat{\text{b}}\flat$-$\hat{\text{a}}$ between the bass line and the soprano. In this particular example, a progression I_c-V-I would be a better choice.
5. There is an occurrence of a false relation in phrase 3 of chorale 'Herzliebster Jesu, was hast du' (No. 78). This results from the assumption used in our algorithm which will firstly determine the chord with on the beat pitch and only look at the other pitch when it is not satisfied with the current solution. This works well for most of the cases but in some cases as in this example it works less well.
6. As a matter of fact, each chorale is unique. Each chorale is different from each

others in its details (e.g. the activities of the parts, the uses of suspension, rhythmic figures). The last example is intentionally selected to show a different texture from the first four examples and to show that with our control model, stylistic information can be explicitly captured and controlled.

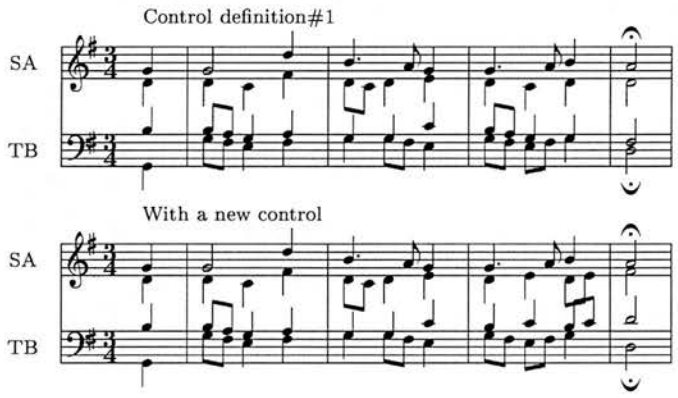
7.2.3.2 Can our control make a difference?

It is our interest that the control definitions should be flexible in order to be able to construct different control definitions for different desired effects. At present, how can we deal with the above points with the current control components in the system? We give some solutions below.

Regarding the first point, we may incorporate a measure control definition which will give a spacing measurement between voices. We can then use this information to select a desired solution. We modify the original version in which we first outline the bass line, then outline the inner voices. The original control is displayed below.

```
.... outlineBass
then outlineInnerVoices
then ....
```

We incorporate the measurement which does not just select the first available answer, but will pick the most desired one from the pool. The harmonisation example below is the result of the first phrase of chorale 'Aus meines Herzens Grunde' (R001) with the new control structure, summarised below after the harmonisation example.




```
n_score outlineBass size 3
then sort_score measure:measure(property(linearOutlineBass))
then n_score outlineInnerVoices size 20
```

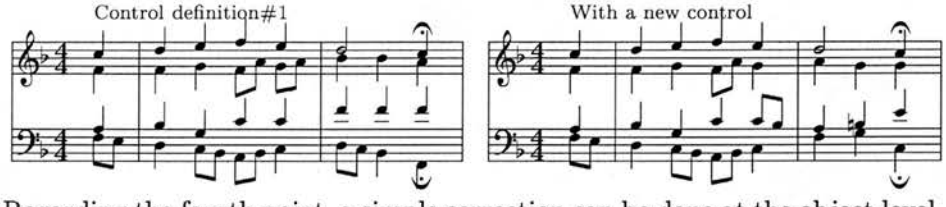
```
        then sort_score (measure:measure(property(spacingInnerVoices))
                         and measure:measure(property(linearOutlineAlto))
                         and measure:measure(property(linearOutlineTenor)))
    then....
```

Regarding the second point, we do not have specific tests or measures implemented for this purpose. However, we could allow multiple possible answers by including non-determinism into the control structure with a **branch** primitive, or by explicitly constructing different control structures for the phrases which have the same outline harmonic structure.

Regarding the third point, we can specify the type of cadence we want with our control definitions. This, of course, gives us a precise control but at the same time makes the control to be specific for a particular harmonisation procedure.

```
%% instead of
.... rule:outlineChord(cadence)
then rule:outlineChord(intro)
then ....
%% we can try
.... rule:outlineChord([cadence(perfect_cadence),key('C major')])
then rule:outlineChord(intro)
then ....
```

We apply this idea to reharmonise phrase 3 of chorale 'Christus, der ist mein Leben' (R006). The original cadence is a plagal cadence in the key of F major, the new version is now a perfect cadence in the key of C major.



Regarding the fourth point, a simple correction can be done at the object level. A logical step would be to check that the key and chord does not conflict with all pitches in the investigated area. However, this is not a panacea as some auxiliary notes may be chromatic.

Regarding the fifth point, we may devise a test to make sure that there is no cadence in the form of *I-V-I* or *I-vii$^{o}_{b}$-I*. This will fix the problem but at the same time not allow the system to do something Bach has done in the very same example

(see Bach's original harmonisation of the same example). Another possibility would be to choose the best bass line from various plausible bass line from different harmonic progressions (e.g. using **n_score** and **sort** primitives as discussed in the first point).

Regarding the final point, we modify our control structure by filling in the outline pitch without any decoration. This is a simple example. A more complex control may be experimented with. For example, we may want the bass line to flow while the alto and tenor parts are less active; or we may want to use more suspensions in certain areas of the chorales, etc. The stylistic applications are still to be further investigated.

7.2.3.3 Comment from experts in the musical qualities

The harmonisation results have been given to Dr John Kitchen and Dr Raymond Monelle of the Faculty of Music, University of Edinburgh, for their comments. In general, both of them think the outputs are quite impressive and interesting. However, there are still many places where the harmonisations are slightly dubious stylistically. We summarise their observations under three headings:

- Inconsistency in performance
- Lack of musical knowledge
- Lacking in musical intention.

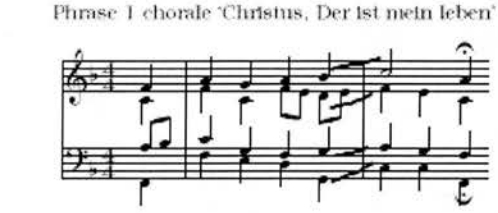
7.2.3.4 Inconsistency in performance

We see inconsistency in performance as a symptom of the lack of deeper knowledge and precise control. This two topics will be discussed after the current topic. Most of computer generated music suffers from this problem. Computers sometimes generate very intelligent and convincing outputs, sometimes with acceptable outputs and sometimes with poor outputs that make us wonder what they are up to. Compare the two bass lines taken from the first phrase and the third phrase of chorale 'Aus meines Herzens Grunde' (No. 1): the third phrase has a better bass line than the first phrase. The choice of cadence is another example, the cadence ii_b^7-V-I on the second phrase of chorale 'Aus meines Herzens Grunde' is very convincing and stylistically correct. However, the cadence in the third phrase in the chorale 'Christus, der ist mein Leben' is less convincing. The cadence in this phrase ends with a plagal cadence IV_b-IV-I in the key of F major which is not wrong. However, both Dr Kitchen and Dr Monelle expect this cadence to be a full closed in C major. The logic here is that since (i) the

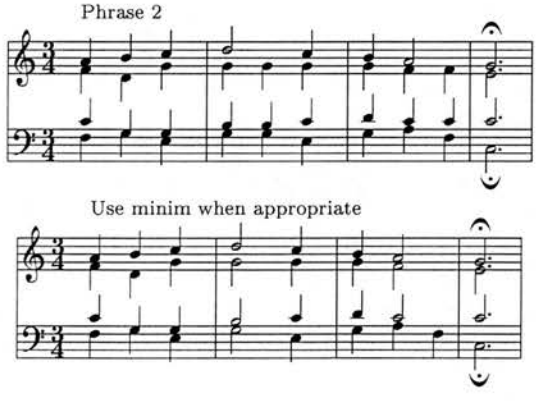
phrase is the third phrase, we may want to get away from the home key and (ii) C major is a near related key which fits very well to the step descending melody (ê, d̂, ĉ) of the phrase. Although the system can write a perfect cadence in C major, it does not choose to do this. Part of the knowledge which could guide the system to do the right thing at the right time is missing and this causes inconsistency in the system's performance. In this example, our object level knowledge tries to outline a cadence in the home key before trying other related key. This is why the plagal cadence in the home key is the selected cadence.

7.2.3.5 Lack of musical knowledge

There is, of course, much knowledge which is not yet implemented in the system due mainly to time constraints. Some of the missing knowledge would be quite easy to implement; for example, the experts suggested that a cadential $\frac{6}{4}$ should not arrive by leaping of all parts in the same direction and landing at the cadential $\frac{6}{4}$ chord.



Another example is from chorale 'Ach Gott, wie manches Herzeleid' (R217); here the minim would be a better choice (in general, the decision depends a lot on the text to be sung with).



It is also suggested that Bach usually employs applied dominants instead of an imperfect cadence *ii*-*V*, that is to raise the third of the supertonic chord to be a major

third (i.e. II-V). The above examples are quite easy to deal with. New knowledge can be easily added locally at the object-level. There are other dimensions which are more sophisticated and the application of knowledge is more global in its nature. Justification in this type of situation has a higher degree of dependency to its circumstances. For examples, some auxiliary notes (e.g. neighbor notes, suspensions) are commented as not appropriate or pointless decorations. We give three examples here. Figure 7.2 shows an example of inappropriate use of a neighbor note and a suggested correction from Dr Monelle. Figure 7.3 and figure 7.4 show examples of a not so meaningful usage of neighbor notes and a suspension as commented by Dr Kitchen.

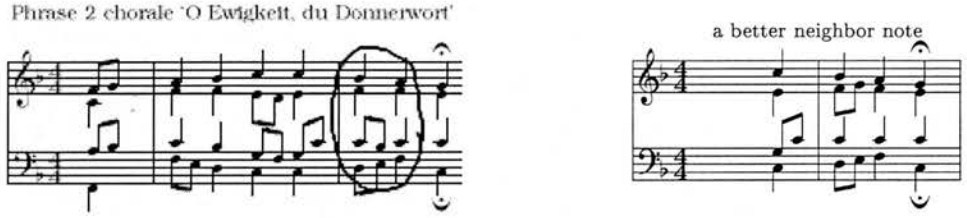


Figure 7.2: A poor neighbor note and a suggested correction (chorale R026)

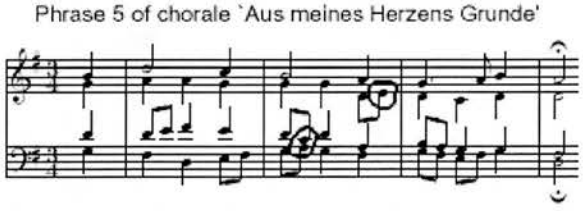


Figure 7.3: Poor neighbor notes (chorale R001)

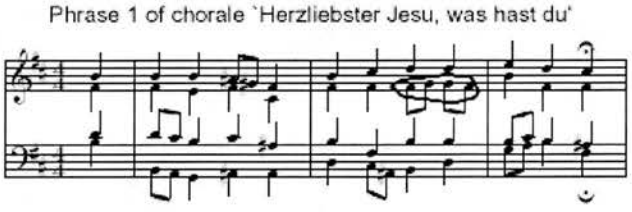


Figure 7.4: Poor suspension usage (chorale R078)

This is a hard problem to deal with (it is also hard for humans too). As a matter of fact, the system is already implemented with heuristics to deal with inappropriate neighbor notes. According to our present domain knowledge, a neighbor note is allowed to be filled only if it forms a continuous step movement with the pitch before or after it. All neighbor notes in our examples comply with this. However, this is apparently not enough. This area is a good one for experimenting with a reflection architecture since the local object-level seems to be less effective in this case.

7.2.3.6 Lacking in musical intention

Perhaps this is the hardest part to model as it is of a higher level knowledge and this type of knowledge is not suitable to be captured as a local object-level knowledge. Firstly, let us make clear what we mean by the term 'musical intention'. We use the term to mean a higher level plan; our control definitions are examples of musical intentions. The musical intention is not a rigid plan. It is the overall concept of how the harmonisation should be carried out. It is usually the case that we cannot foresee all harmonisation details in advance. Therefore, to maintain the overall coherence, local activities must be justified so that they do not jeopardise the coherence of the overall shape. It may be possible to have an intensive rigorous type of knowledge at the local level which can foresee other consequences at a global level, but this type of knowledge is more natural at the meta-level. This area is another good candidate for experimenting with a reflection architecture.

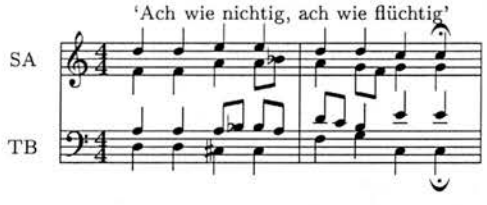


Figure 7.5: Choices of key (chorale R048)

Both experts point out that, in many phrases, our system seems not to know what key it is in. As an example, Dr Kitchen comments that the harmonisation in the phrase 5 of chorale 'Ach wie nichtig, ach wie flüchtig' (reproduced here in figure 7.5) does not work well since the melody does not have any accidental. So the harmonisation in the key of D minor at the beginning and modulate to C major at the cadence may

not be a very good decision. This scenario mostly happens in the middle of the piece since we encourage the system to explore other key areas in the body of the chorale. At present, modulating passages are still constructed as a local problem in a local area and this could result in a strange choice of key. The choice of plagal cadence discussed earlier is also in this category. Dr Kitchen also points out that sometimes the bass line seems to lack direction (see chorale 'Aus meines Herzens Grunde' figure 7.6).

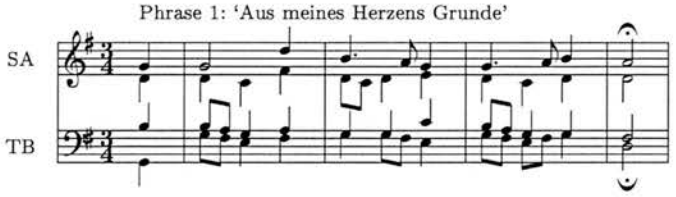


Figure 7.6: Lacking direction in the bass (chorale R001)

In this example, the bass line displays too much repetition. It seems that our heuristic of maintaining the bass line direction does not work well in this case (see page 84 for details of the domain knowledge in this topic).

7.3 Evaluations of Control Provided in the System

Generally, systems cannot perform better than what is in their domain knowledge but they can perform worse than expected if they are not equipped with proper controls. In this section, we discuss the following two topics.

- Experiment with various control ideas
- Weakness of the system

7.3.1 Experiment with various Control ideas

In section 7.2.3.2, we have illustrated how can we use the existing control components to express our compositional intention. In this section, we elaborate the issue further by giving harmonisation outputs which are harmonised with control specifically designed for them. There are three main strategies that we rely on in composing a control structure.

- The context in control definitions.

- The order of unfolding processes.
- The incorporation of tests and measures to guide search.

7.3.1.1 The context in control definitions

One possible strategy in harmonising a chorale would be to start with the last phrase then do the first repeated section of the chorale and then do the rest of the chorale. This is a sound strategy since the context of the chorales is closely related to the closing and the opening of the chorales. The control structure may be constructed in this way:

```
%% the last phrase can be directly specified, for example
rule:selectPhrase(outlineHarmonicPlan,phrase-6)

%% or by using filter to make the control structure more general
filter  rule:selectPhrase(outlineHarmonicPlan)
with    test:property( phraseID,lastPhrase )
```

It is important to devise atomic control definitions in such a way that they carry appropriate context suitable for our reasoning strategy.

7.3.1.2 The order of unfolding processes

After the complete harmonic outline of all the phrases is obtained, we may choose to outline the bass line before other voices since the contour of the bass line and the melody have a closer relationship than the soprano and other voices. The control structure may be constructed in various ways according to the way the process is conceived. We may want to complete the whole bass line before we start filling the inner voices.

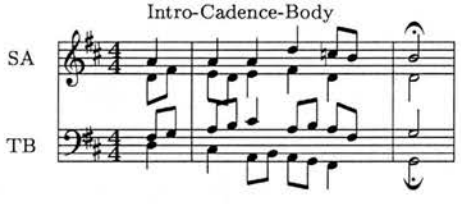
```
definition( outlineVoices,
      %% fill the whole bass line first
      rule:initialisePhrase( outlineVoices )
      then ( backtrackRepeat
             rule:outlineBass( transition )
             until  test:property(status(outlineBass),allFilled))
      then rule:closePhrase( outlineVoices )
      %% fill the inner voices
      then rule:initialisePhrase( outlineVoices )
      then ( backtrackRepeat
             ( rule:outlineAlto( transition )
               then rule:outlineTenor( transition ) )
```

```
            until  test:property(status(outlineTenor),allFilled))
    then rule:closePhrase( outlineVoices ) ).
```

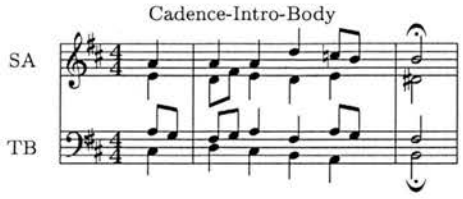
Alternatively, we may fill the bass, then the alto then the tenor before moving to the next step. These methods explore the search space in completely different orders.

```
definition( outlineVoices,
      rule:initialisePhrase( outlineVoices )
      then ( backtrackRepeat
           ( rule:outlineBass( transition )
             then rule:outlineTenor( transition )
             then rule:outlineAlto( transition ) )
           until test:property(status(outlineAlto),allFilled))
      then rule:closePhrase( outlineVoices ) ).
```

The two examples below are the harmonisation of phrase 5 of the chorales *O Gott, du frommer Gott* No. 312 [Riemenschneider, 1941]. The choice of chorale is arbitrary, the choice of phrase is deliberate (i.e. it should be something in the middle of the chorale since harmonic structure in the middle of the chorale has more flexibility in terms of key). It does not have to end in the home key. We chose the middle part for this example since it is is freer in the harmonic structure. In the first example, the harmonic progression is determined by deciding on the start then the cadence and then the rest in the body. In the second example, the harmonic progression is determined by deciding on the cadence before the start and the body.

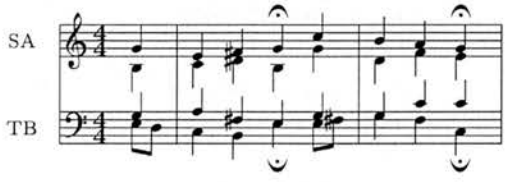


In the above example (an intro-cadence-body order), the harmonic progression starts off in the key of D major before closing with perfect cadence in the key of G major. In the second example below (a cadence-intro-body order), the harmonic progression starts off in the same D major key but closing in imperfect cadence in the key of E minor.

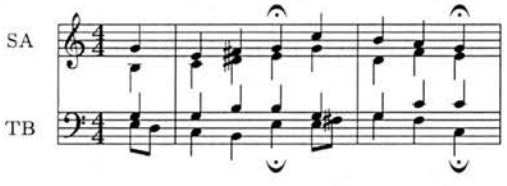


7.3.1.3 The incorporation of Tests and Measures to guide search

The following example is an instance of the harmonisation of phrase 8 and 9 of Bach's chorales: *Ich dank' dir, Gott für Wohltat* No. 223 [Riemenschneider, 1941].



Suppose we want to move the tenor part up higher, we may wish to control this by constraining the output with a test *'property(lowerBound(tenor),[5,nat,3])'*. Below is the result of both the eighth and ninth phrase when the test is added into the control structure.



This means that the solution with the tenor part below  would be filtered out. The same property may be diluted as a soft constraint with a measure. In this case, solutions with the tenor line below the bass clef E3 will not be absolutely rejected. The output will be ranked and the least flawed one would be selected. This allows more flexibility in applying heuristics which are not just black or white, but grey.

7.3.1.4 Chorale: Aus meines Herzens Grunde R001

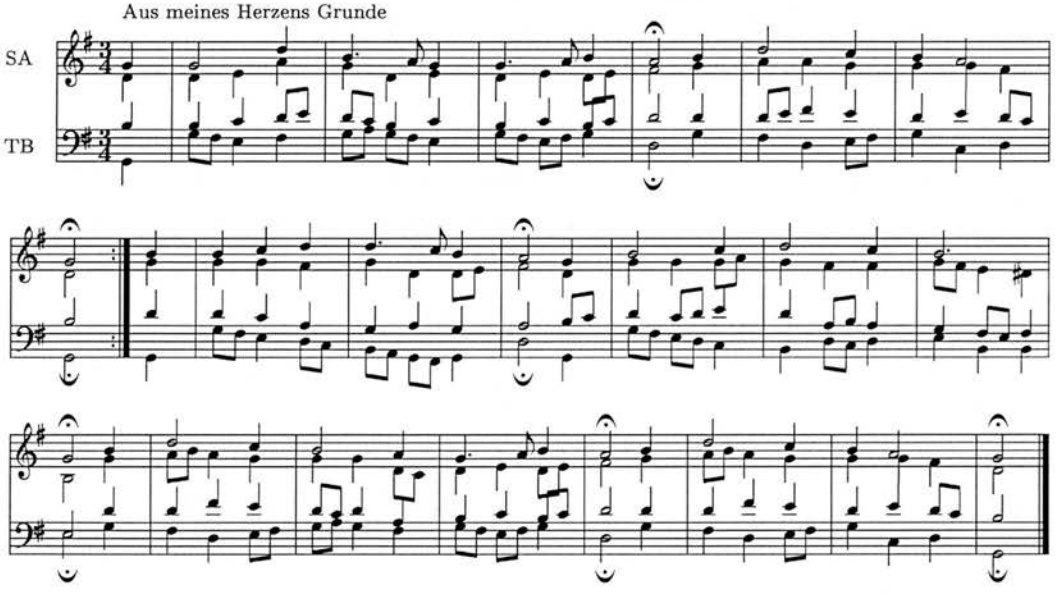
We reharmonised the chorale 'Aus meines Herzens Grunde' again with a new control structure (r001.def).[3]. In the new control, we employ two strategies in trying to increase

[3]Control definitions r001.def, r006.def, r080.def, r217.def and r350.def are available online at http://www.dai.ed.ac.uk/homes/somnukpa/esc/

the distance between the bass voice and the tenor voice. The first strategy is by using the **n_score** and the **sort** primitives. The second strategy is to use the **filter–with** primitive.

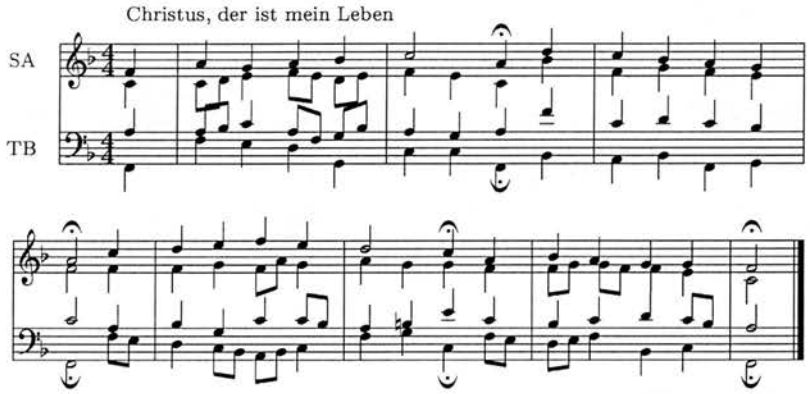
```
%% use n_score and sort in selecting the bass line
n_score outlineBass size 3
then sort_score measure:measure( property(linearOutlineBass) )
then ...
%% use n_score and sort in selecting the alto and the tenor lines
n_score outlineInnerVoices size 20
then sort_score ( measure:measure( property(spacingInnerVoices) )
                  and measure:measure( property(linearOutlineAlto) )
                  and measure:measure( property(linearOutlineTenor) ) )
then ...
%% use filter--with
filter outlineInnerVoices
with test:property(lowerBound(tenor),[5,nat,3])
then ...
```

We also try to introduce some variation between phrase 2 and the last phrase which share the same melody. In this example, we simply swap the order of the filling-in of the alto and the tenor voices.



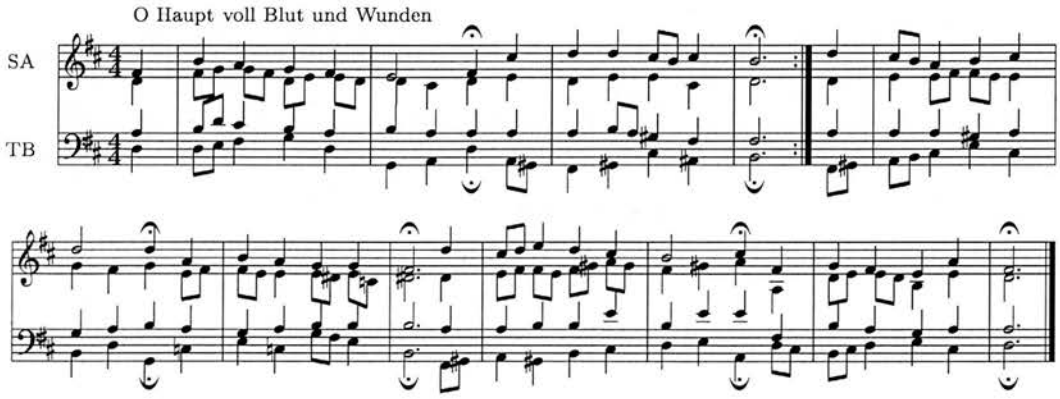
7.3.1.5 Chorale: Christus, der ist mein Leben R006

This harmonisation is different from the first version in some details. Here, we incorporate tests and measures to guide search. The plagal cadence in the third phrase in our first version (see page 174) has been changed into the perfect cadence in the key of C major in this new harmonisation. There are some minor changes in the first phrase as well.



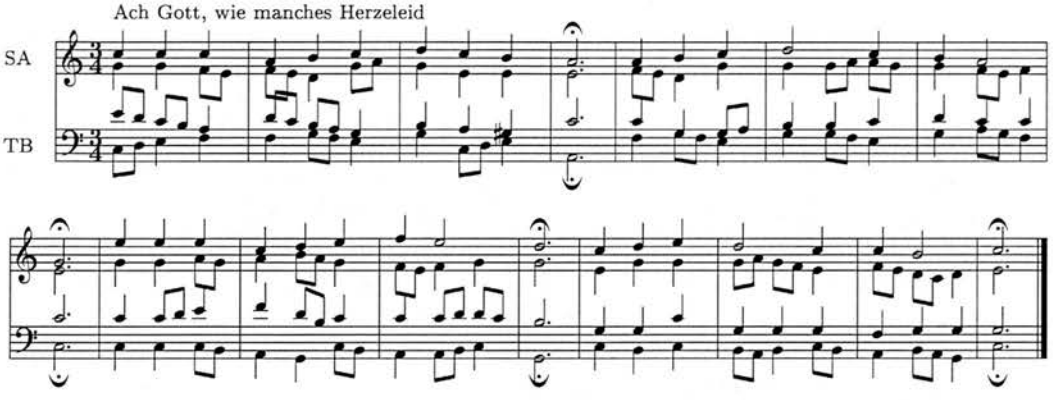
7.3.1.6 Chorale: O Haupt voll Blut und Wunden R080

In this example, we outline the cadence in the penultimate phrase with a perfect cadence in the key of A major. The filling in the alto voice has been modified from the original control definition #3. This gives an overall effect of more dissonant seconds in the alto voice (see the first version in page 175).

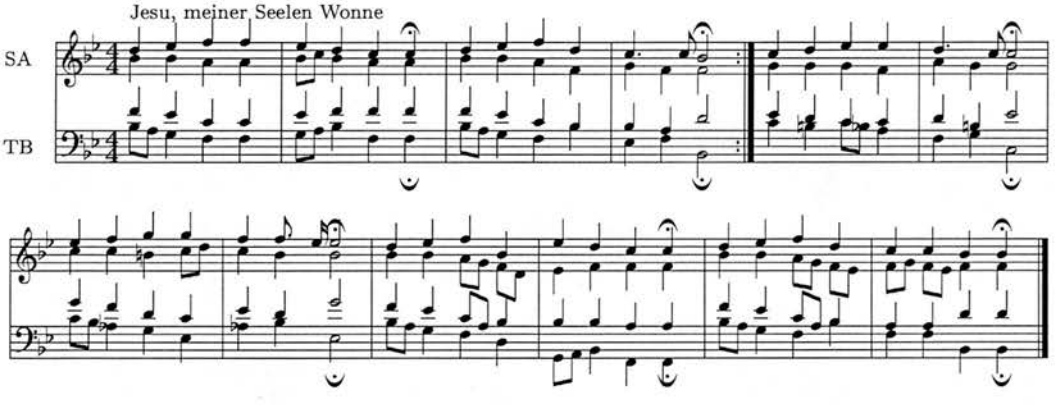


7.3.1.7 Chorale: Ach Gott, wie manches Herzeleid R217

The control definition r217.def gives a different texture to the chorale (see previous harmonised example in page 176).



7.3.1.8 Chorale: Jesu, meiner Seelen Wonne R350



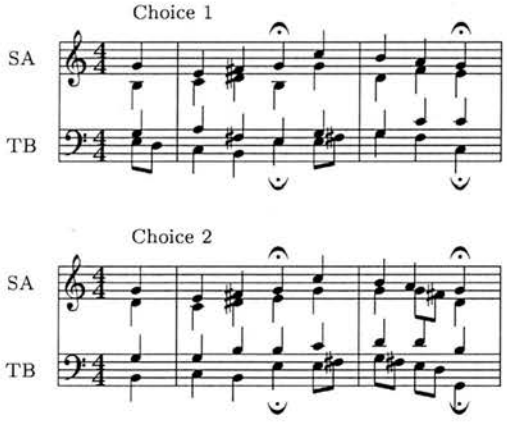
In this last example, we outline cadence in the third phrase with a perfect cadence in the key of Eb major. The control definitions used for harmonising these examples (r001.def, r006.def, r080.def, r217.def and r350.def) are available online at `http://www.dai.ed.ac.uk/homes/somnukpa/esc/`. These examples illustrate that many good plausible harmonisation outputs are plausible with different control structures; and with explicitly structured control model, new control structures are composed at the meta-level with ease and flexibility.

7.3.2 Weaknesses of the system

Although the previous three examples are constructed for each different chorale, we still feel that we do not have a grasp of control at the desired level. The overall result of the above examples shows an improvement in their quality. However the level of control is at a very coarse grain size. At the current stage, if we want to use more suspensions in some voices, we can construct our control structure for those voices in such a way that a suspension is tried before other options. This gives an overall effect of more uses of suspension, but not the detailed control over how it should be used.

As another example, the bass line in phrase 1 is still showing the symptom of 'lacking in direction' as put by Dr Kitchen. We will need more atomic control definitions to address this 'lacking in direction' symptom. Many plausible strategies could be employed to help creating direction in the work. Revision can be in the direction of the bass line (e.g. to establish a bass line movement of tonic–dominant, tonic–dominant–tonic); in the melodic structure of the soprano, alto, tenor and bass line (e.g. use the same rhythmic or melodic idea in other part to create homogeneity).

Another example below illustrates that certain control feature is not possible at the current stage. In this demonstration, we take phrase 8 and 9 of chorale 'Ich dank dir, Gott für Wohltat' (No. 223) and show different plausible outputs with different control structures (see details of the control structure used in this example in appendix A).



In this case, assuming we prefer one output over the other, how can we express this. We know that we have the **branch–with** primitive, which will allow us to express our preference at branching points. However, currently we do not have appropriate

means (i.e. appropriate atomic measures) for the task.

Generally, we experience inadequacy of atomic tests and atomic measures rather than inadequacy of atomic rules. The atomic rules always give plausible solutions, while the atomic tests and atomic measures provide us with a means to have more control over a set of plausible solutions. This does not mean that atomic rules are perfect; it would be even better (in our opinion) to have these atomic rules at finer grain sizes but still maintain the context of compositional processes (i.e. we do not want these atomic definitions to go down at the same level as an object language).

7.4 Conclusion

Harmonising Bach chorales is a hard AI problem, comparable to natural language understanding. In language, a sentence can be grammatically correct but meaningless, or grammatically correct and meaningful but not appropriate for the situation. The same parallels are observed in music. In this class of problems, solutions must be both correct (syntactically well-formed) and appropriate to the context (semantically sound). There seems to be no fixed universal algorithms for this class of problem.

For a complex problem like chorale harmonisation, search control with heuristics is a more effective approach than rigid algorithm since the search can be guided and controlled. Unstructured approaches using artificial neural networks and GAs do not seem appropriate. Search control in these two approaches are from abstract domain dependent control knowledge (e.g. the notions of cross over and mutation in the GAs, the notion of weight between neuron connections in ANNs). In the unstructured search approach, the link between control knowledge and its effects is not clear. The structured approach (e.g. symbolic AI search) shows a clear link between control knowledge and its effects. This strong link enhances the understanding of the control knowledge and the traversal path in the search space. We view this as a very important factor especially in performing a partial search in a search space of this size (see chapter 3 for a comparison between the GAs and the rule based approach).

Existing systems based on structured approach (which are capable of producing reasonable harmonisation output like CHORAL) are less than transparently structured. To produce reasonable chorale harmonisations in Bach style, the knowledge base cannot be small. Knowledge base of this size which are not modularly and explicitly constructed are usually impenetrable and are very hard to generalise from and hence,

very hard to extend or expand.

Generally, human composers exercise powerful heuristics in their problem solving techniques (e.g. the way problems are looked at, broken down and approached). We capture these heuristics in our harmonisation process (expressed with our control language). Search in the space of plausible harmonisation outputs is explicitly guided using our control language. Two main components in the control language are atomic control definitions and control primitives. The atomic control definitions are modular; this allows us to modify earlier control structure for a desired behaviour with flexibility. The control primitives allow us to combine different control definitions together. Data streams from different control definitions are linked by the control primitives. The stream of solutions is controlled (e.g. constrained, re-prioritised) according to heuristics we wish to express in the control structure.

In this thesis, we have shown that our meta-interpreter and our control language offer a powerful means for controlling search using heuristics. Our control language provides a natural means to capture musical knowledge in modular and musically understandable forms. The control structures can be easily modified since their components are modular. The effect of the modification observed in the harmonisation results are easy to understand since atomic control definitions are musically understandable.

At this stage in our research, the system displays many attractive features (i.e. control definitions are easy to construct, modify). However, it is still far from being able to offer flexibility at the level that reflects the composition process in humans. There are many areas with many facets which could be extended from the current research point. We shall discuss this in the next chapter.

7.4.1 Summary of our Contributions

This research has extensively investigated the modelling of a harmonisation process using explicit and modular control definitions. We suggest that our problem solving approach offers many attractive features in constructing a machine model of complex musical expertise. We summarise our contributions below:

- Modular structure of knowledge components promotes reusability of knowledge content as well as flexibility in search control. The same set of atomic definitions can yield many different control structures (see section 7.2.3.2 and section 7.3.1).
- In general, our control structures are simple, easy to understand and easy to

construct. Users can modify the control structures with ease and flexibility as illustrated in chapter 7. It is clear that compositional processes prove to be very useful heuristics and complement the missing part of musical knowledge (which is not apparent in the finished product).

- As for the appropriateness of control primitives and atomic control definitions, this is a less successful area. In our opinion, we believe the nature of both of these control components are dependent upon each other. That is, the shape of one affects the shape of the other. There is some control we could not express with the present control components (see page 191). We see this as mainly due to the lack of appropriate atomic test and atomic measure control definitions. Improvement can be made in this area by expanding the existing control libraries.
- At present, the system relies on the backtracking mechanism in Prolog, although we have promoted flexibility of control so that control structures can be constructed in such a way that backtracking can be minimised. There are many situations in which a more powerful control to deal with unnecessary backtracking would be preferred (see page 163).
- We have successfully achieved our goal (stated in chapter 5) regarding the representation framework. The system supports transformation of music materials to different data structures for different computational purposes. This is manifested in the encapsulated I/O for all control definitions (see chapter 6, page 131).
- Our harmonisation outputs display convincing stylistic harmonisation passages (see harmonisation examples in chapter 7). We see this as resulting from a better search control and we believe the explicit control is a promising approach to this problem.

Chapter 8

Conclusions and Further work

Introduction

In this chapter, we discuss further work extendable from the current stage of our research. Apart from an immediate refinement of the current implementation (e.g. adding utilities for user friendliness), further work extending from the current research point could be carried out in various interesting focused areas. We propose the following possibilities:

- Further work in interaction between human and machines.
- Further work in data structure design.
- Further work in control primitives and atomic control definitions.
- Further work in reflection at a meta-level.

8.1 Further Work

8.1.1 Interaction between Human and Machines

At the moment, interaction with the program is driven by the program structure. The interaction point is predefined in the atomic rules, atomic tests and atomic measures. Human and machine interaction can be further expanded to the level at which humans can work interactively with machines to produce harmonisation output. We believe that this is a very important concept as it could change our common conception of machines. When the computer is not just either an efficient but dull helper, or a very clever helper but deaf, we should find synergy in combination.

Improvement in this area is not alien to our system architecture. Reflection architecture (as illustrated in section 8.1.4.1) enhances expressiveness in reasoning. Modular structure of knowledge components promotes communication between knowledge components which is essential in a cooperative environment. At present, users can ask simple 'Why?' and 'How?' questions. The system answers these questions using a script which is hard coded in each control definition. The script can access run time information (e.g. answering a 'how' question with search path information, answering a 'why' question by including run time information in the script).

For the system to be able to provide precise and useful information, there must be a kind of reasoning process behind it. We propose a further extension of our architecture to include a reflection module which will be discussed in section 8.1.4 in this chapter. We hope this will facilitate us in accepting a more complex input from users and vice versa, to provide elaborate and detailed information for users. That is, interaction is not limited to a simple 'why' or 'how' question but can involve more sophisticated information.

More details in the solution state would be very beneficial. Consider our harmonisation process when the current state does not lead to the solution state. In this case, the program backtracks in order to exploit other parts of the search space. However, chronological backtracking can be a timely and expensive process. This is not a new problem. The CHORAL system addresses it with an intelligent backtracking system. We, at present, deal with it using an explicitly structured control model which allows us to explicitly construct definitions in the most suitable way for a particular problem in order to reduce backtracking. However, if more information is known we may be able to dramatically improve the efficiency of the system.

8.1.2 Refining data structure design

At the current stage, the control definitions operate on either a single *score* or a list of *scores*. The score is a very high level abstract data type, and in this experiment we decided to adhere to it since it allows the flexibility to access all information (i.e. every atomic control definition can access all the information available in the score). For example, there are two ways to fill in inner voices in our system. We could use *outlineInnerVoices* which will fill in the alto and tenor of a phrase in one go. This is like a black box. Control of this operation is modifiable only at the object level.

On the other hand we could use *outlineAlto(transition)* and *outlineTenor(transition)*, which allow better flexibility in applying control, since control can now be applied at each transition step (i.e. depending on the grain size of a transition step in the implementation, which is at each crotchet in our implementation).

In the latter approach users must construct appropriate tests and measures which are hidden at the object-level inside the *outlineInnerVoices*. However, the flexibility gained also comes with a computational cost. To reduce the computational cost, we propose one alternative here. Since part of the computation expense occurs as a result of bulky score structure being passed around, every control definition must unwrap some kind of syntactic wrapping paper before gaining access to useful information. If only useful data for a specific task is packed together then this part of the overhead could be greatly reduced since the meta-interpreter can directly access this information. We show how this is done in the meta-interpreter below (see meta-interpreter on page 124).

```
% solve( +ControlDef, ?Cinp, -Cout, ?Input, -Output )
% ControlDef : Control definition
% Cinp : Input control data
% Cout : Output control data
% Input : Input data
% Output : Output data
% TypeInput : Input data for a Type
% TypeOutput : Output data for a Type
   solve( Type:ControlDef,Cinp,Cout,Input,Output ) ←
            ¬ memberchk( Type,[rule,test,measure] ) ∧
            definition( Type:ControlDef,ControlDefs ) ∧
            retrieveType( Type,Input,TypeInput ) ∧
            solve( ControlDefs,Cinp,Cout,TypeInput,TypeOutput ) ∧
            updateType( Type,TypeOutPut,Output )
```

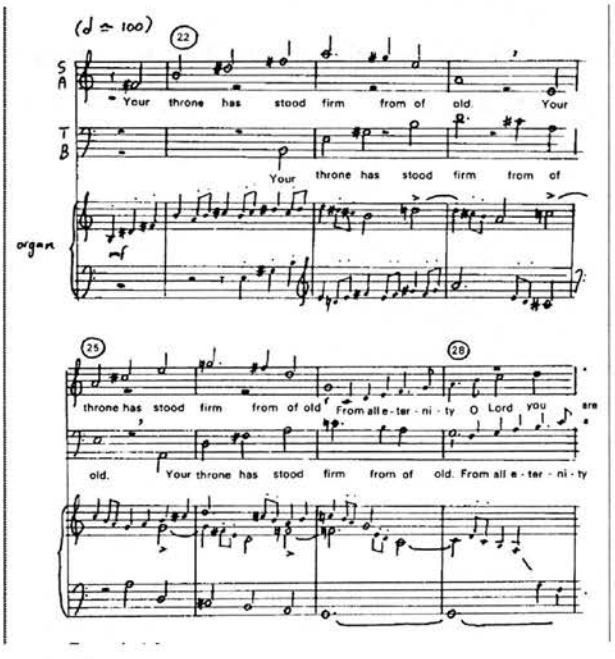
New control primitives and control definitions dealing with new data types could be devised. This gives us a lot of rooms for improving the performance of the system (e.g. by improving control primitives, control definitions and different data types). For example, if we perform an operation on a list of data, we must test if all the data in the list has been processed. On the other hand, if a control definition directly operates on the list data structure, the test is not necessary since the data structure implicitly provides this information.

8.1.3 Refining Control Components

In chapter 7, we pointed out some limitations of our control components (see page 191). We mentioned earlier that the designs of control primitives and control definitions are dependent, and a shifting in granularity of the I/O data can open a whole new path for new choices of control components (as shown in the previous section).

In this section, let's look at a broader view of the refinement in control components. There is plenty of room for further development in this area. Sloboda [1985] suggests that observations of composers at work and asking composers to think aloud could contribute significant information regarding the 'compositional process'. This information would be very useful for refining our control components (i.e. both control primitives and control definitions). He also gives an example of his own protocol of a 'twentieth-century English church music' composition. We shall quote some of them below:

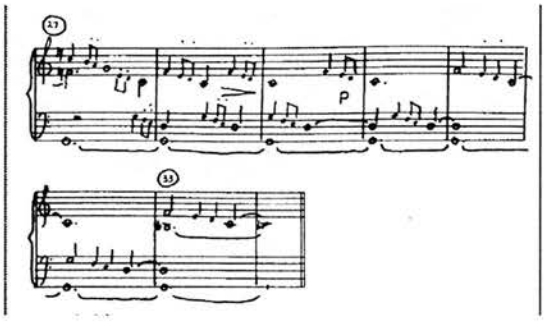
> "How am I to continue at bar 29? The next verse is going to be quieter and slower. A theme is needed for 'From all eternity'. The tentative one where composition broke down before didn't seem right, partly because it was over too quickly."



Excerpt from Sloboda [1985], page 126

In real music, these are the type of questions which constantly appear in a composer's mind. It is a strategic plan to get the music to express the desired tones and colours with resources available in that musical idiom. Sloboda comes up with a solution below:

> "I decided to progress the accompaniment so that it slows down. It occurs to me that the word 'eternity' can be represented by a repetitive circular motif" (he referred to the top organ part in bars 23-27). "So I try to use it to slow down. The writing of bars 23-27 then becomes routine. A B♭ is added at bar 33 on the intuition that harmonic movement is needed by now."



Excerpt from Sloboda [1985], page 128

From the above protocol, it should be evident that it is not a trivial task to design generic atomic definitions which could manifest the thought as quoted above. At the current stage, our control definitions could capture composition thinking and illustrate composition processes. The grain size of composition ideas is still very crude. This is still unexplored territory. We expect to see the atomic definitions devised in such a way that they capture only relevant information specifying the composer's intention in fine detail. However, we do not expect to see tasks which are not relevant to composition thoughts to appear at this level, for example: counter control, data type control (e.g. counting 1 to 100, is the value as integer?, is the variable grounded?).

8.1.4 Reflection at the Meta-level

Perhaps the most exciting and interesting area in modelling human expertise in a machine is the modelling of the reflection ability; that is, to enable the machine to reason out its own actions. Smith [1985] gives the reflection hypothesis which states that:

> "In as much as a computational process can be constructed to reason about an external world in virtue of comprising an ingredient process (interpreter) formally manipulating representations of the world, so too a computational process could be made to reason about itself in virtue of comprising an ingredient process (interpreter) formally manipulating representation of its own operation and structures."

As in a general control situation, where the control could be mixed with the program at the object level, in the same way, the information used in the reflection could also be mixed at the object level. Here, we use the same strategy by constructing the reflection theory at the meta-level. The same context mapping mechanism is employed to provide a connection between the object-level domain and the meta-level reflection. This endows the program with the ability to reason out its own actions. The next thing that we will need is to construct a new meta-theory describing the object theory. Generally speaking, the new meta-theory may discuss the following information:

1. Data structure information: the meta theory may discuss the data structure (e.g. the shape of data structure as in the *PRESS* system [Bundy and Welham, 1981])
2. Search structure information: the meta theory may discuss the search structure. This could be solution path information (e.g. a plan; history of how the search space is traversed) or search control information (e.g. how the control is applied).
3. Solution state information: the meta theory may discuss the object theory in a meta language (e.g. as the control knowledge in *NEOMYCIN*).

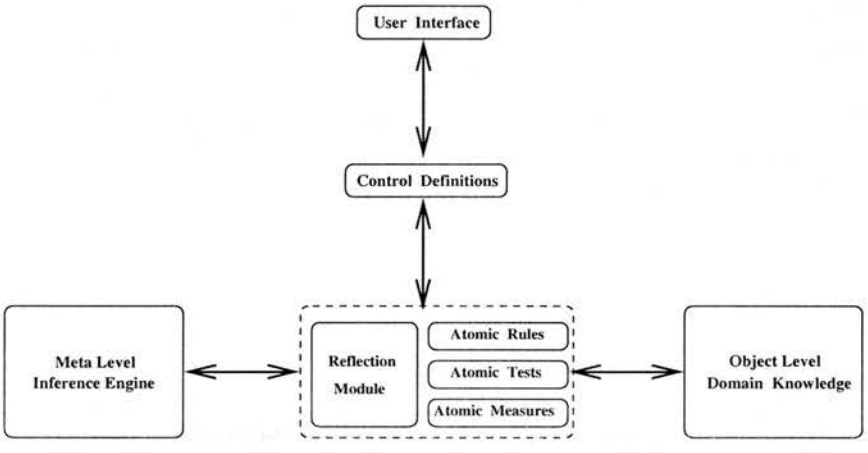
Our current implementation does not fully exploit the data structure information. Information regarding the search structure and the solution state is explored to a certain extent. The solution state information is mapped onto the meta-level context for constructing explicit control. The solution path (a plan) is recorded for answering the 'How' question. A new meta-theory for reflection could extensively explore the above information sources and enhance system performance, for example, explanation can be elaborated with more details. We will discuss this dimension later on in this section.

8.1.4.1 Reflection architecture

The reflection principle is the statement of a relation between a theory and its meta-theory [Weyhrauch, 1985]. Bowen and Kowalski [1982] see the effectiveness of promoting communication between the two levels. They give the linking rules which are

known as upward and downward reflection principles. The upward rule allows the object-level to communicate its object-theory to the meta-level. The downward rule allows the meta-level to communicate its meta-theory to the object-level. Although the meta-level has a less complex search space, many object-level problems can be more naturally and more efficiently solved at the object-level [Bowen and Kowalski, 1982].

We suggest that both reflection mechanisms could be useful for our further work. We propose the reflection architecture framework below.



This is basically the same architecture as in the one illustrated in chapter 6. The explanation module is now seen as the reflection module (as a matter of fact, our explanation module is implemented with the blackboard architecture). The basic idea is very simple: the atomic definitions will perform the tasks of posting and retrieving information from the blackboard and inspecting this information. With this architecture, the *downwardReflection* can communicate the meta-theories to the object-level, and vice versa the *upwardReflection* can communicate the object-theories to the meta-level. The meta-level knowledge posted to the blackboard is a meta-theory representing useful control information (i.e. information regarding data structure, search structure and solution state).

8.1.4.2 Applications using reflection

One crucial ingredient in effective control is the reflective ability. An explicit representation of the domain and control knowledge enhances the ease of knowledge maintenance, and allows the system to reason about the control in isolation from the domain. The

system could reflect on what has been done so far. This could be reasoning about the shape of data structures; the plan applied so far; the contents of the solution and what type of control is going to be applied next; or if the present control structure would lead to a solution. This meta-level reasoning requires a new appropriate meta theory. We discuss possible areas where further improvement on reflection at the meta-level may be carried out: (i) enhanced explanations and (ii) critics.

- **Enhanced explanation**: Explanation to 'Why' and 'How' questions in expert systems generally exploit the search path information. The template for this explanation is normally in a simple form such as (modified from [Bratko, 1990]):

  How?
  goal(a,b) was derived by rule R_x from
  p(a,c) was derived by rule R_y
  and
  q(b,d) was derived by rule R_z from
  r(c) was in the knowledge-based
  ...

  Why? (why do you want to know a?)
  because:
  I can use a to investigate b by rule R_a and
  I can use b to investigate c by rule R_b and
  ...
  I can use y to investigate z by rule R_y and
  z is your original question.

  The above explanation framework is appealing from its simplicity. However, it lacks expressiveness in explanations. We took a different approach in constructing an explanation system. Our explanation system is implemented at each atomic definition. We believe this is a good strategy and the current explanation system could be enhanced with information from both the meta-level and the object-level which are available from the reflective architecture. The fuller the information provided, the clearer the transparency of the solution state. This is very useful and is essential for further development in tutoring and teaching purposes.
- **Critics**: Currently, if the current state does not lead to the solution state, the program backtracks in order to exploit other parts of the search space. However, blind chronological backtracking can be very expensive. It would be a very useful

facility if the system could reason and suggest plausible problem areas during run-time. The information would be very useful in the composition of a new control definition, or for a very ambitious mind, to alter the control definitions at run-time. In the proof planning technique [Ireland, 1992; Ireland and Bundy, 1996] the proof planner has *critics* which capture common patterns of failures in the proof plan and suggest how to patch the failed proof plan. We can use the same intuition in our reflective mechanism.

The ideas in sections 8.1.1, 8.1.2, 8.1.3 and 8.1.4 should be very interesting topics for future research.

Appendix A

Control Definitions

This appendix gives control definition examples. These examples aim to illustrate different control features constructed from available control definitions. We summarise control features in each control definitions below:

A.1 Control definition #1

This control definition is designed in a way that it is general to most harmonisation problem. It produces a reasonable output in most cases. It approaches the harmonisation task with a simple plan:

- The system starts from the first phrase and progresses to the last phrase.
- In each phrase, the system completes the harmonisation task by:
  -Determining the harmonic progression
  -Determining the bass line
  -Outlining the inner voices
  -Filling in detail decorations

```
%%
definition( view_melody,
        interactive( rule:analyse(inputMelody) )
        then
        rule:display(choraleMelody) ).
%%
definition( view_harmony,
        view_melody
        then repeat( rule:selectPhrase(outlineHarmonicPlan)
```

```
                    and test:constrain( exposedSecond,bass)
                    and test:constrain( contrarySecond,bass)
                    and test:constrain( oddSlotDissonant,bass) ) )
            then
            ( filter fillinTenor
              with ( test:constrain( oddSlotUnison  )
                    and test:constrain( cadenceLeadingNote,tenor )
                    and test:constrain( seventhPrepared,tenor )
                    and test:constrain( rhythmicPattern,tenor  )
                    and test:constrain( tooManyRepeatedNotes,tenor )
                    and test:constrain( oddSlotCongestion,tenor )
                    and test:constrain( xover,tenor )
                    and test:constrain( consecutiveFifthOctave,tenor)
                    and test:constrain( parallelSecond,tenor)
                    and test:constrain( exposedSecond,tenor)
                    and test:constrain( contrarySecond,tenor)
                    and test:constrain( oddSlotDissonant,tenor) ) )
            then
            ( filter fillinAlto
              with ( test:constrain( cadenceLeadingNote,alto )
                    and test:constrain( seventhPrepared,alto )
                    and test:constrain( rhythmicPattern,alto  )
                    and test:constrain( tooManyRepeatedNotes,alto )
                    and test:constrain( oddSlotCongestion,alto )
                    and test:constrain( xover,alto )
                    and test:constrain( consecutiveFifthOctave,alto)
                    and test:constrain( parallelSecond,alto)
                    and test:constrain( exposedSecond,alto)
                    and test:constrain( contrarySecond,alto)
                    and test:constrain( oddSlotDissonant,alto)
                    and test:constrain( rhythmicPattern,global  ) ) )
            then rule:deselectPosition )
        until test:property(status(ornamentFillin),allFilled)  ).
%%
definition( fillinBass,
        rule:selectVoice( bass )
        then ( (if test:property( state(bass),normal)
              then_do applyNormal)
```

```
                or (if test:property( state(bass),suspension)
                   then_do applySuspension)
                or (if test:property( state(bass),descPassing)
                   then_do applyDescPassing) )
        then rule:deselectVoice( bass ) ).
%%
definition( fillinTenor,
        rule:selectVoice( tenor )
        then ( (if test:property( state(tenor),normal)
               then_do applyNormal)
               or (if test:property( state(tenor),suspension)
                  then_do applySuspension)
               or (if test:property( state(tenor),descPassing)
                  then_do applyDescPassing) )
        then rule:deselectVoice( tenor ) ).
%%
definition( fillinAlto,
        rule:selectVoice( alto )
        then ( (if test:property( state(alto),normal)
               then_do applyNormal)
               or (if test:property( state(alto),suspension)
                  then_do applySuspension)
               or (if test:property( state(alto),descPassing)
                  then_do applyDescPassing) )
        then rule:deselectVoice( alto ) ).
%%
definition( applyNormal,
            rule:fill(passingNote)
            or rule:fill(descPassingI4Bass)
            or rule:fill(descPassingI3Bass)
            or rule:fill(skipI3BassDown)
            or rule:fill(neighborNote)
            or rule:fill(suspension)
            or rule:fill(outlinePitch)
            or rule:fill(skipI3I4)
            or rule:fill(neighborDescPassing)
            or rule:fill(thirdDescPassing )
            or rule:fill(skipI4Alto)
```

```
            or rule:fill(skipI4Tenor)
            or rule:fill(escapeNote)
            or rule:fill(escapeSuspensionNote)
            or rule:fill(neighborSuspension) ).
%%
definition( applySuspension,
            rule:fill(holdSuspension)
            or rule:fill(suspension)
            or rule:fill(resolveStepDown)
            or rule:fill(decorateSuspension)
            or rule:fill(accentedPassingNote) ).
%%
definition( applyDescPassing,
            rule:fill(resolveStepDown)
            or rule:fill(accentedPassing)
            or rule:fill(suspension) ).
```

A.2 Control definition #2

This control definition is a variation of the previous control definition (i.e. Control definition #1). It is designed to harmonise with a plain texture (i.e. a simple crotchet movement). We have modified the fill in decoration part in control definition #1 to fill only the outline pitch in the new control structure.

```
%%
definition( view_melody,
        interactive( rule:analyse(inputMelody) )
        then rule:display(choraleMelody) ).
%%
definition( view_harmony,
        view_melody
        then repeat( rule:selectPhrase(outlineHarmonicPlan)
             then ( filter outline_phrase_harmonic_plan
                    with ( test:constrain( globalHarmonicPlanOutline )
                           and test:constrain( phraseHarmonicPlanOutline ) ) ) )
        then rule:display(harmonicOutline) ).
%%
definition( outline_phrase_harmonic_plan,
           rule:initializePhrase(outlineHarmonicPlan)
           then rule:outlineChord(cadenceA)
           then rule:outlineChord(intro)
           then rule:outlineChord(body)
           then rule:closePhrase(outlineHarmonicPlan) ).
%%
definition( view_outline,
        view_melody
        then repeat( rule:selectPhrase( outlineVoices )
             then ( filter outline_phrase_harmonic_plan
                    with ( test:constrain( globalHarmonicPlanOutline )
                           and test:constrain( phraseHarmonicPlanOutline ) ) )
             then rule:display(partsOutline)
             then outlineVoices
             then rule:display(partsOutline) ) ).
```

```
%%
definition( outlineVoices,
        outlineBass
        then rule:display(partsOutline)
        then outlineInnerVoices  ).
%%
definition( outlineBass,
          rule:initializePhrase( outlineVoices )
          then rule:outlineBass( cadence )
          then ( filter rule:outlineBass( intro )
                 with test:constrain( phraseConsecutiveFifthOctave,bass ) )
          then rule:outlineBass( body )
          then rule:closePhrase( outlineVoices ) ).
%%
definition( outlineInnerVoices,
          rule:initializePhrase( outlineVoices )
          then ( filter rule:outlineInnerVoices
                 with test:constrain(phraseConsecutiveFifthOctave) )
          then rule:closePhrase( outlineVoices ) ).
%%
definition( harmonise,
         view_melody
         then
         fillinChoralStyle  ).
%%
definition( fillinChoralStyle,
         repeat( rule:selectPhrase(fillinProcess)
           then ( filter outline_phrase_harmonic_plan
                  with ( test:constrain( globalHarmonicPlanOutline )
                         and
                         test:constrain( phraseHarmonicPlanOutline ) ) )
                then outlineBass
                then rule:display(harmonisationResult)
                then outlineInnerVoices
                then rule:display(harmonisationResult)
                then rule:initializePhrase(fillinProcess)
                then fillinView
                then rule:closePhrase(fillinProcess)
```

```
                then rule:display(harmonisationResult) )
          then rule:export2MIDI ).
%%
definition( fillinView,
          backtrackRepeat
                ( rule:selectPosition
                  then ( filter fillinBass
                       with ( test:constrain( exposedFifthOctave,bass )
                            and test:constrain( directExposedOctave,bass )
                            and test:constrain( consecutiveSkips,bass )
                            and test:constrain( cadenceLeadingNote,bass )
                            and test:constrain( rhythmicPattern,bass  )
                            and test:constrain( xover,bass )
                            and test:constrain( consecutiveFifthOctave,bass)
                            and test:constrain( parallelSecond,bass)
                            and test:constrain( exposedSecond,bass)
                            and test:constrain( contrarySecond,bass) ) )
                  then ( filter fillinTenor
                       with ( test:constrain( cadenceLeadingNote,tenor )
                            and test:constrain( seventhPrepared,tenor )
                            and test:constrain( rhythmicPattern,tenor  )
                            and test:constrain( xover,tenor )
                            and test:constrain( consecutiveFifthOctave,tenor)
                            and test:constrain( parallelSecond,tenor)
                            and test:constrain( exposedSecond,tenor)
                            and test:constrain( contrarySecond,tenor) ) )
                  then ( filter fillinAlto
                       with ( test:constrain( cadenceLeadingNote,alto )
                            and test:constrain( seventhPrepared,alto )
                            and test:constrain( rhythmicPattern,alto  )
                            and test:constrain( xover,alto )
                            and test:constrain( consecutiveFifthOctave,alto)
                            and test:constrain( parallelSecond,alto)
                            and test:constrain( exposedSecond,alto)
                            and test:constrain( contrarySecond,alto)
                            and test:constrain( rhythmicPattern,global ) ) )
                  then rule:deselectPosition )
          until test:property(status(ornamentFillin),allFilled)  ).
```

```
%%
definition( fillinBass,
          rule:selectVoice( bass )
          then (if test:property( state(bass),normal)
               then_do rule:fill(outlinePitch) )
          then rule:deselectVoice( bass ) ).
%%
definition( fillinTenor,
          rule:selectVoice( tenor )
          then (if test:property( state(tenor),normal)
               then_do rule:fill(outlinePitch) )
          then rule:deselectVoice( tenor ) ).
%%
definition( fillinAlto,
          rule:selectVoice( alto )
          then (if test:property( state(alto),normal)
               then_do rule:fill(outlinePitch) )
          then rule:deselectVoice( alto ) ).
```

A.3 Control definition #3

This control definition is also a variation of the control definition #1.

```
%%
definition( view_melody,
        interactive( rule:analyse(inputMelody) )
        then rule:display(choraleMelody) ).
%%
definition( view_harmony,
        view_melody
        then repeat( rule:selectPhrase(outlineHarmonicPlan)
             then ( filter outline_phrase_harmonic_plan
                    with ( test:constrain( globalHarmonicPlanOutline )
                           and test:constrain( phraseHarmonicPlanOutline ))))
        then rule:display(harmonicOutline) ).
%%
definition( outline_phrase_harmonic_plan,
           rule:initializePhrase(outlineHarmonicPlan)
           then rule:outlineChord(cadenceA)
           then rule:outlineChord(intro)
           then rule:outlineChord(body)
           then rule:closePhrase(outlineHarmonicPlan) ).
%%
definition( view_outline,
        view_melody
        then repeat( rule:selectPhrase( outlineVoices )
             then ( filter outline_phrase_harmonic_plan
                    with ( test:constrain( globalHarmonicPlanOutline )
                           and test:constrain( phraseHarmonicPlanOutline )))
             then rule:display(partsOutline)
             then outlineVoices
             then rule:display(partsOutline) ) ).
%%
definition( outlineVoices,
         outlineBass
         then rule:display(partsOutline)
         then outlineInnerVoices  ).
```

```
%%
definition( outlineBass,
        rule:initializePhrase( outlineVoices )
        then rule:outlineBass( cadence )
        then ( filter rule:outlineBass( intro )
               with test:constrain( phraseConsecutiveFifthOctave,bass ) )
        then rule:outlineBass( body )
        then rule:closePhrase( outlineVoices ) ).
%%
definition( outlineInnerVoices,
        rule:initializePhrase( outlineVoices )
        then ( filter rule:outlineInnerVoices
               with test:constrain(phraseConsecutiveFifthOctave) )
        then rule:closePhrase( outlineVoices ) ).
%%
definition( harmonise,
         view_melody
         then fillinChoralStyle  ).
%%
definition( fillinChoralStyle,
         repeat( rule:selectPhrase(fillinProcess)
            then ( filter outline_phrase_harmonic_plan
                   with ( test:constrain( globalHarmonicPlanOutline )
                          and test:constrain( phraseHarmonicPlanOutline ) ) )
            then first_score( outlineBass )
            then rule:display(harmonisationResult)
            then outlineInnerVoices
            then rule:display(harmonisationResult)
            then rule:initializePhrase(fillinProcess)
            then fillinView
            then rule:closePhrase(fillinProcess)
            then rule:display(harmonisationResult) )
            then rule:export2MIDI ).
%%
definition( fillinView,
         backtrackRepeat
           ( rule:selectPosition
             then ( filter fillinBass
```

```
        with ( test:constrain( exposedFifthOctave,bass )
               and test:constrain( directExposedOctave,bass )
               and test:constrain( consecutiveSkips,bass )
               and test:constrain( cadenceLeadingNote,bass )
               and test:constrain( rhythmicPattern,bass  )
               and test:constrain( melodicPatternXYXY,bass )
               and test:constrain( tooManyRepeatedNotes,bass )
               and test:constrain( oddSlotCongestion,bass )
               and test:constrain( xover,bass )
               and test:constrain( consecutiveFifthOctave,bass)
               and test:constrain( parallelSecond,bass)
               and test:constrain( exposedSecond,bass)
               and  test:constrain( contrarySecond,bass)
               and  test:constrain( oddSlotDissonant,bass) ) )
then ( filter fillinAlto
        with ( test:constrain( cadenceLeadingNote,alto )
               and test:constrain( seventhPrepared,alto )
               and test:constrain( rhythmicPattern,alto  )
               and test:constrain( tooManyRepeatedNotes,alto )
               and test:constrain( oddSlotCongestion,alto )
               and test:constrain( xover,alto )
               and test:constrain( consecutiveFifthOctave,alto)
               and test:constrain( parallelSecond,alto)
               and test:constrain( exposedSecond,alto)
               and test:constrain( contrarySecond,alto)
               and test:constrain( oddSlotDissonant,alto)
               and test:constrain( rhythmicPattern,global ) ) )
then ( filter fillinTenor
        with ( test:constrain( oddSlotUnison  )
               and test:constrain( cadenceLeadingNote,tenor )
               and test:constrain( seventhPrepared,tenor )
               and test:constrain( rhythmicPattern,tenor  )
               and test:constrain( tooManyRepeatedNotes,tenor )
               and test:constrain( oddSlotCongestion,tenor )
               and test:constrain( xover,tenor )
               and test:constrain( consecutiveFifthOctave,tenor)
               and test:constrain( parallelSecond,tenor)
               and test:constrain( exposedSecond,tenor)
```

```
                        and test:constrain( contrarySecond,tenor)
                        and test:constrain( oddSlotDissonant,tenor) ) )
            then rule:deselectPosition )
        until test:property(status(ornamentFillin),allFilled)  ).
%%
definition( fillinBass,
        rule:selectVoice( bass )
        then ( (if test:property( state(bass),normal)
               then_do applyNormalB)
               or (if test:property( state(bass),suspension)
                   then_do applySuspension)
               or (if test:property( state(bass),descPassing)
                   then_do applyDescPassing) )
        then rule:deselectVoice( bass ) ).
%%
definition( fillinTenor,
        rule:selectVoice( tenor )
        then ( (if test:property( state(tenor),normal)
               then_do applyNormalT)
               or (if test:property( state(tenor),suspension)
                  then_do applySuspension)
               or (if test:property( state(tenor),descPassing)
                  then_do applyDescPassing) )
        then rule:deselectVoice( tenor ) ).
%%
definition( fillinAlto,
        rule:selectVoice( alto )
        then ( (if test:property( state(alto),normal)
               then_do applyNormalA)
               or (if test:property( state(alto),suspension)
                  then_do applySuspension)
               or (if test:property( state(alto),descPassing)
                  then_do applyDescPassing) )
        then rule:deselectVoice( alto ) ).
%%
definition( applyNormalB,
           rule:fill(passingNote)
           or rule:fill(outlinePitch)
```

```
            or rule:fill(descPassingI4Bass)
            or rule:fill(descPassingI3Bass)
            or rule:fill(skipI3BassDown)
            or rule:fill(neighborNote)
            or rule:fill(suspension)
            or rule:fill(neighborDescPassing)
            or rule:fill(thirdDescPassing )
            or rule:fill(neighborSuspension) ).
%%
definition( applyNormalT,
            rule:fill(passingNote)
            or rule:fill(outlinePitch)
            or rule:fill(suspension)
            or rule:fill(escapeSuspensionNote)
            or rule:fill(neighborSuspension)
            or rule:fill(neighborNote)
            or rule:fill(skipI3I4)
            or rule:fill(neighborDescPassing)
            or rule:fill(thirdDescPassing )
            or rule:fill(skipI4Tenor)
            or rule:fill(escapeNote) ).
%%
definition( applyNormalA,
            rule:fill(passingNote)
            or rule:fill(suspension)
            or rule:fill(neighborNote)
            or rule:fill(outlinePitch)
            or rule:fill(escapeSuspensionNote)
            or rule:fill(neighborSuspension)
            or rule:fill(skipI3I4)
            or rule:fill(neighborDescPassing)
            or rule:fill(thirdDescPassing )
            or rule:fill(skipI4Alto)
            or rule:fill(escapeNote) ).
%%
definition( applySuspension,
            rule:fill(holdSuspension)
            or rule:fill(suspension)
```

```
            or rule:fill(resolveStepDown)
            or rule:fill(decorateSuspension)
            or rule:fill(accentedPassingNote) ).
%%
definition( applyDescPassing,
            rule:fill(resolveStepDown)
            or rule:fill(accentedPassing)
            or rule:fill(suspension) ).
```

A.4 Control definition #4

This control definition illustrates that each phrase may be applied with a different control structure. Here, we determine the harmonic progression of the first, second and third phrase. Then we outline the bass line of the first phrase, and outline both the bass line and the inner voices of the third phrase. The ordering of task is one strategy that can be employed for controlling search.

```
%%
definition( view_melody,
        interactive(rule:analyse(inputMelody))
        then
        rule:display(choraleMelody) ).
%%
definition( view_harmony,
        view_melody
        then rule:selectPhrase(outlineHarmonicPlan,phrase-1)
        then outline_phrase_harmonic_plan
        then rule:display(harmonisationResult)
        then rule:selectPhrase(outlineHarmonicPlan,phrase-2)
        then outline_phrase_harmonic_plan
        then rule:display(harmonisationResult)
        then rule:selectPhrase(outlineHarmonicPlan,phrase-3)
        then outline_phrase_harmonic_plan
        then rule:display(harmonisationResult) ).
%%
definition( outline_phrase_harmonic_plan,
          rule:initializePhrase(outlineHarmonicPlan)
          then rule:outlineChord(cadence)
          then rule:outlineChord(intro)
          then rule:outlineChord(body)
          then rule:closePhrase(outlineHarmonicPlan) ).
%%
definition( view_outline,
          view_harmony
          then rule:selectPhrase(outlineVoices,phrase-1)
          then outlineVoicesPhrase1
          then rule:selectPhrase(outlineVoices,phrase-3)
```

```
          then outlineVoicesPhrase3 ).
%%
definition( outlineVoicesPhrase1,
          outlineBass ).
%%
definition( outlineVoicesPhrase3,
          outlineBass
          then outlineInnerVoices ).
%%
definition( outlineBass,
           rule:initializePhrase( outlineVoices )
           then rule:outlineBass( cadence )
           then rule:outlineBass( intro )
           then rule:outlineBass( body )
           then rule:closePhrase( outlineVoices )
           then rule:display(harmonisationResult) ).
%%
definition( outlineInnerVoices,
           rule:initializePhrase( outlineVoices )
           then rule:outlineInnerVoices
           then rule:closePhrase( outlineVoices )
           then rule:display(harmonisationResult) ).
```

A.5 Control definition #5

This control definition illustrates how measures are used to guide the search to a desired solution. The controls *Def1* and *Def2* are designed to outline the bass voice.

```
%%
definition( def1,
          interactive( rule:analyse(inputMelody) )
          then rule:selectPhrase(fillinProcess,phrase-1)
          then ( filter outline_phrase_harmonic_plan
               with ( test:constrain( globalHarmonicPlanOutline )
                    and test:constrain( phraseHarmonicPlanOutline ) ) )
          then outlineBassDef1
          then rule:display(harmonisationResult) ).
%%
definition( outline_phrase_harmonic_plan,
          rule:initializePhrase(outlineHarmonicPlan)
          then rule:outlineChord(cadence)
          then rule:outlineChord(intro)
          then rule:outlineChord(body)
          then rule:closePhrase(outlineHarmonicPlan) ).
%%
definition( outlineBassDef1,
          rule:initializePhrase( outlineVoices )
          then ( backtrackRepeat rule:outlineBass( transition )
                 until test:property(status(outlineBass),allFilled) )
          then rule:closePhrase( outlineVoices ) ).
%%
definition( def2,
          interactive( rule:analyse(inputMelody) )
          then rule:selectPhrase(fillinProcess,phrase-1)
          then ( filter outline_phrase_harmonic_plan
               with ( test:constrain( globalHarmonicPlanOutline )
                    and test:constrain( phraseHarmonicPlanOutline ) ) )
          then outlineBassDef2
          then rule:display(harmonisationResult) ).
%%
definition( outlineBassDef2,
          rule:initializePhrase( outlineVoices )
```

```
            then outlineBass2
            then rule:closePhrase( outlineVoices ) ).
%%
definition( outlineBass2,
            rule:outlineBass( cadence )
            then rule:outlineBass( intro )
            then rule:outlineBass( body ) ).
```

Appendix B

Harmonisation Examples

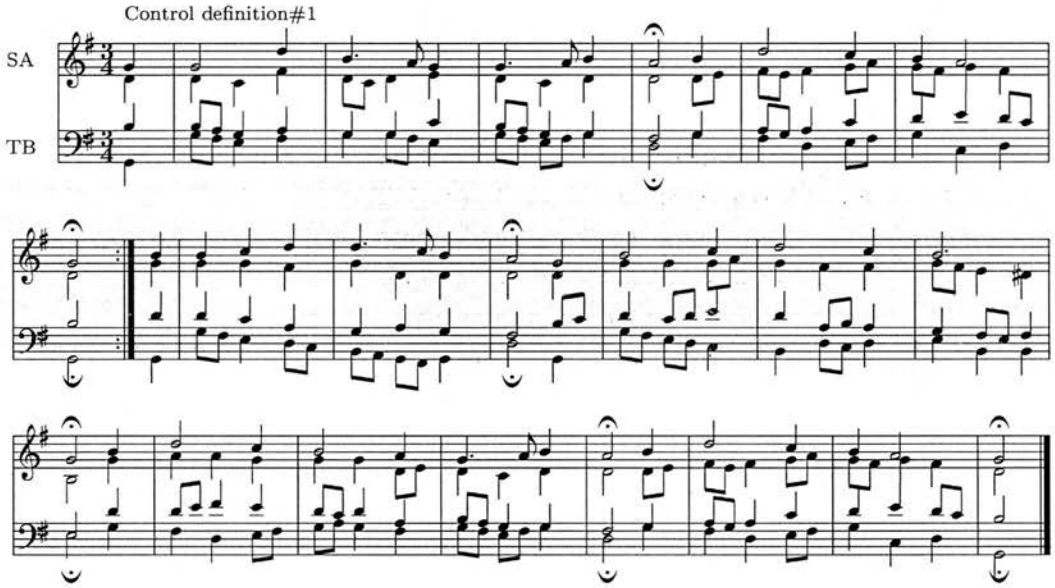
This appendix includes harmonisation examples from our systems. The relevant control structures can be looked up in appendix A. The table below summarises all the examples given in this appendix. Many editions of Bach's chorales exist in the repertoire. This work uses the Riemenschneider [1941] 371 Harmonised Chorales as our main resource. The harmonisation examples here are produced with three different control definitions, however, the choice of the control is arbitrary except in chorale No. 217 where we want to make a point about the texture of chorales. Control definitions r001, r006, r080, r217 and r350 are also being composed specifically for chorale No. 1, 6, 80, 217 and 350 respectively.

B.1 Examples from Our System

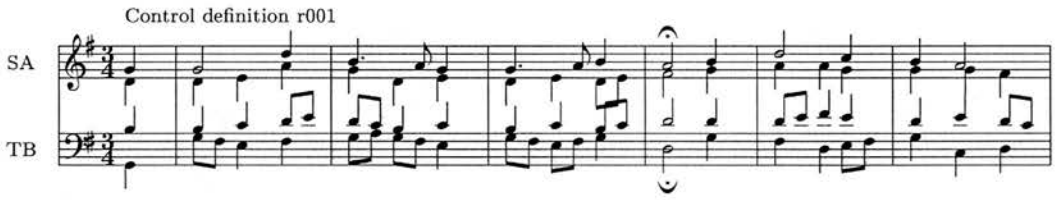
Chorales No.	Chorales Name	Control Definitions
1	Aus meines Herzens Grunde	1, r001
6	Christus, der ist mein Leben	1, r006
9	Ermuntre dich, mein schwacher Geist	3
26	O Ewigkeit, du Donnerwort	1
44	Machs mit mir, Gott, nach deiner Gut	3
48	Ach wie nichtig, ach wie fluchtig	1
78	Herzliebster Jesu, was hast du	1
80	O Haupt voll Blut und Wunden	3, r080
88	Helft mir Gotts Gute preisen	3
125	Allein Gott in der Hoh sei Ehr	3
151	Meinen Jesum lab ich nicht, Jesus	1
159	Als der gutige Gott	1

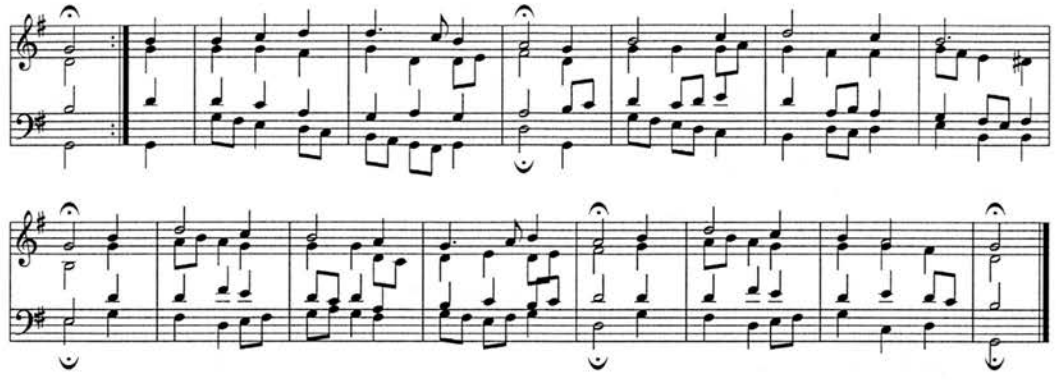
175	Jesus, meine Zuversicht	3
217	Ach Gott, wie manches Herzeleid	2, r217
223	Ich dank dir, Gott, fur all Wohltat	1
228	Danket dem Herren, denn er ist sehr freundlich	3
274	O Ewigkeit, du Donnerwort	3
304	Auf meinen lieben Gott	1
310	Machs mit mir, Gott, nach deiner Gut	3
324	Jesu, meine Freude	3
330	Nun danket alle Gott	3
340	Befiehl du deine Wege	1
350	Jesu, meiner Seelen Wonne	r350

B.1.1 R001 Aus meines Herzens Grunde

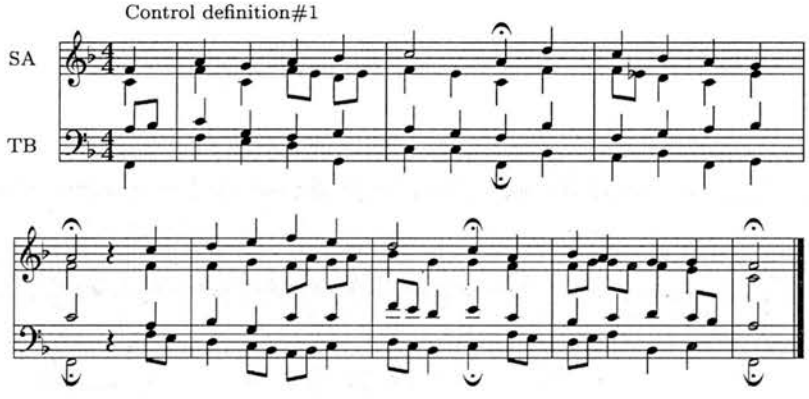


B.1.2 R001 Aus meines Herzens Grunde

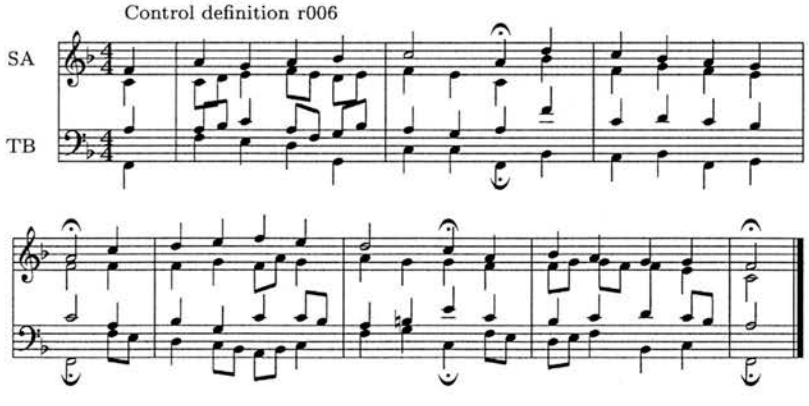




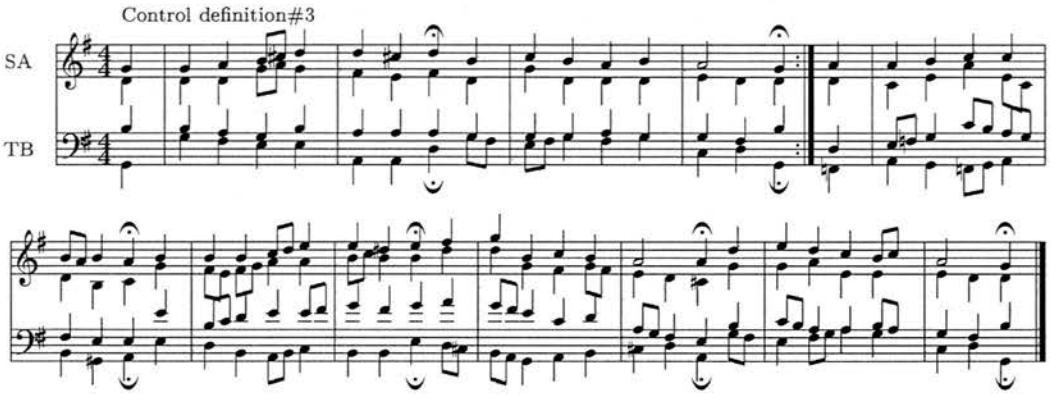
B.1.3 R006 Christus, der ist mein Leben



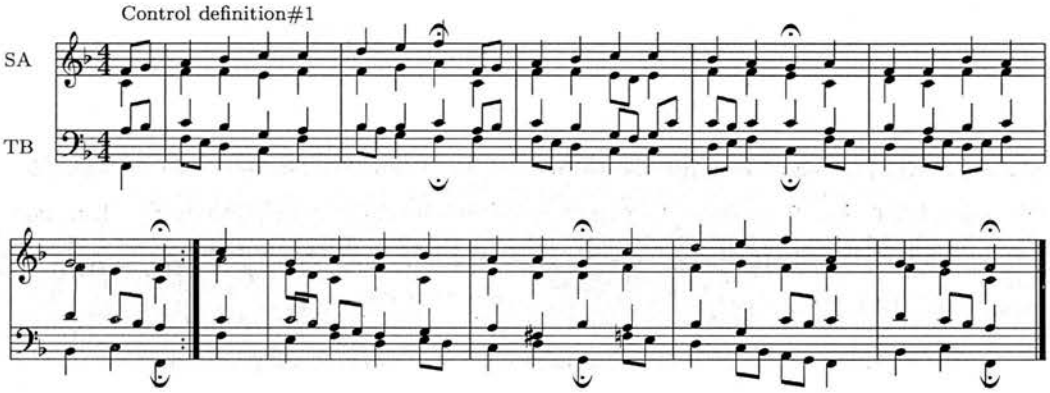
B.1.4 R006 Christus, der ist mein Leben



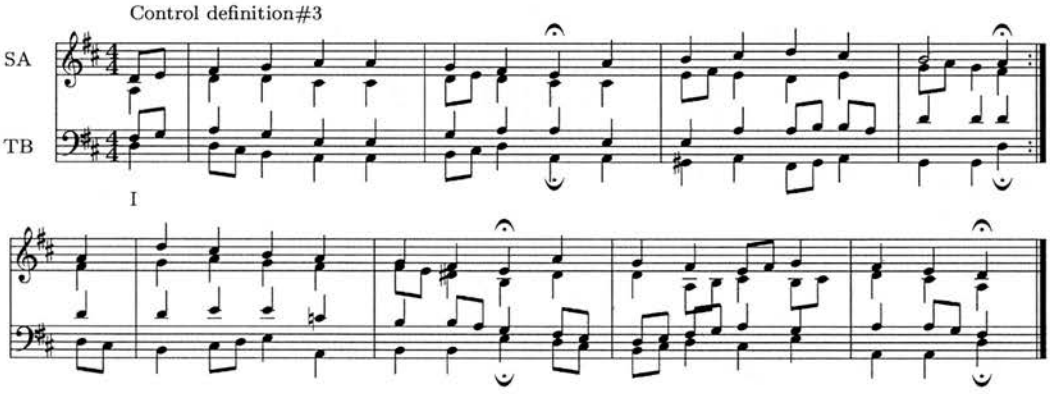
B.1.5 R009 Ermuntre dich, mein schwacher Geist



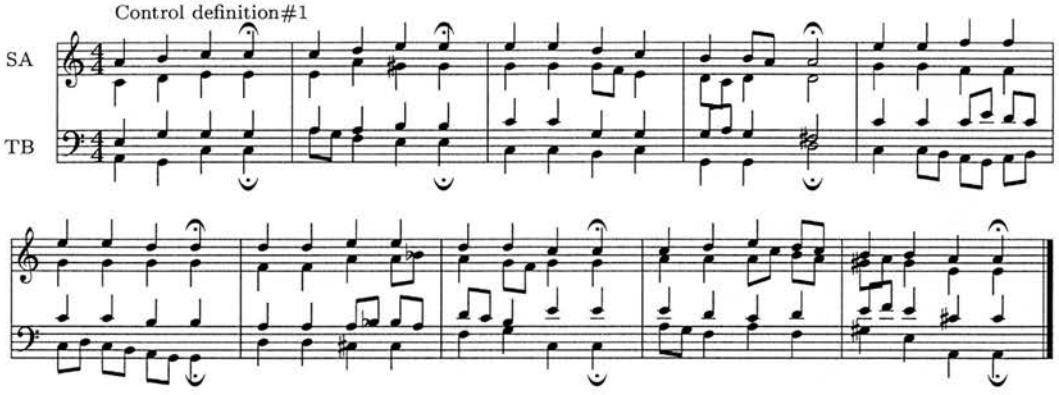
B.1.6 R026 O Ewigkeit, du Donnerwort



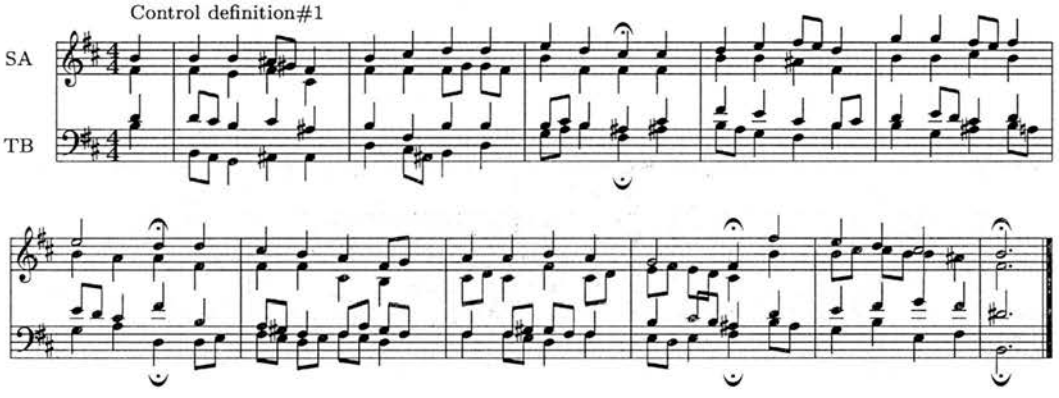
B.1.7 R044 Machs mit mir, Gott, nach deiner Gut



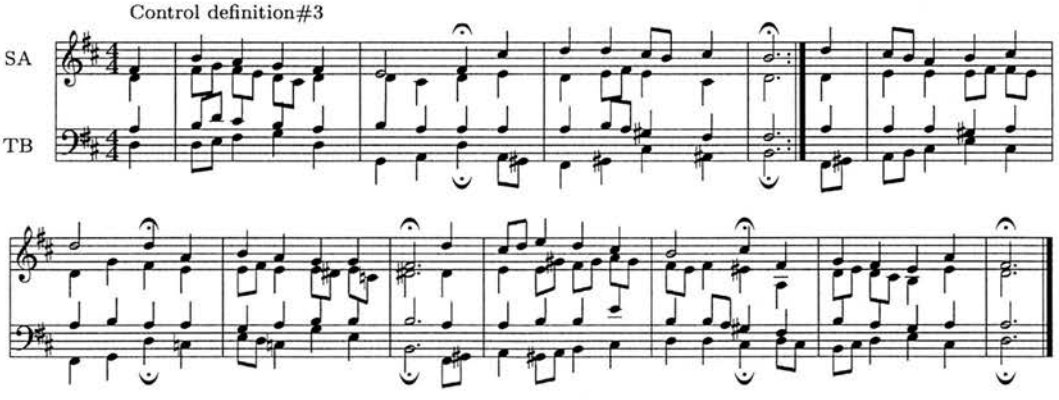
B.1.8 R048 Ach wie nichtig, ach wie fluchtig



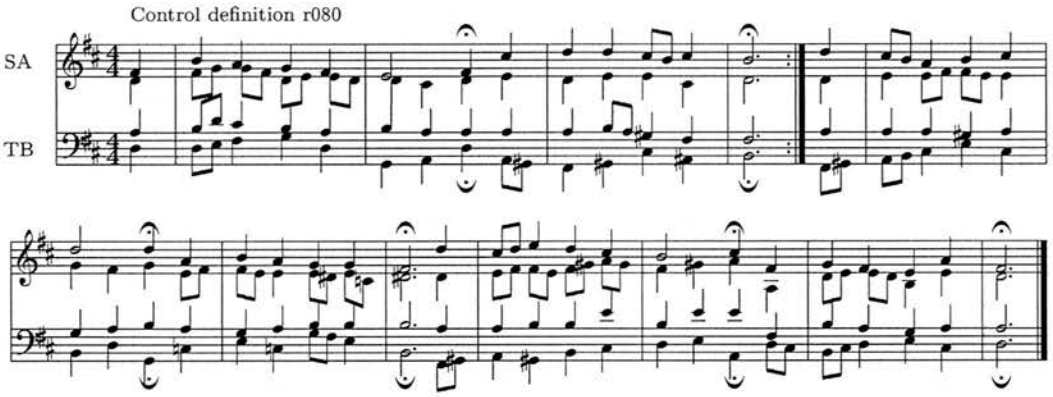
B.1.9 R078 Herzliebster Jesu, was hast du



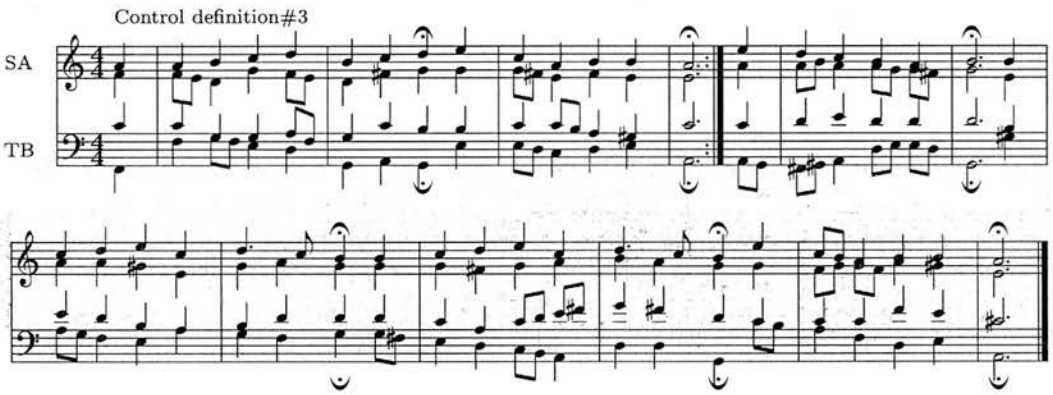
B.1.10 R080 O Haupt voll Blut und Wunden



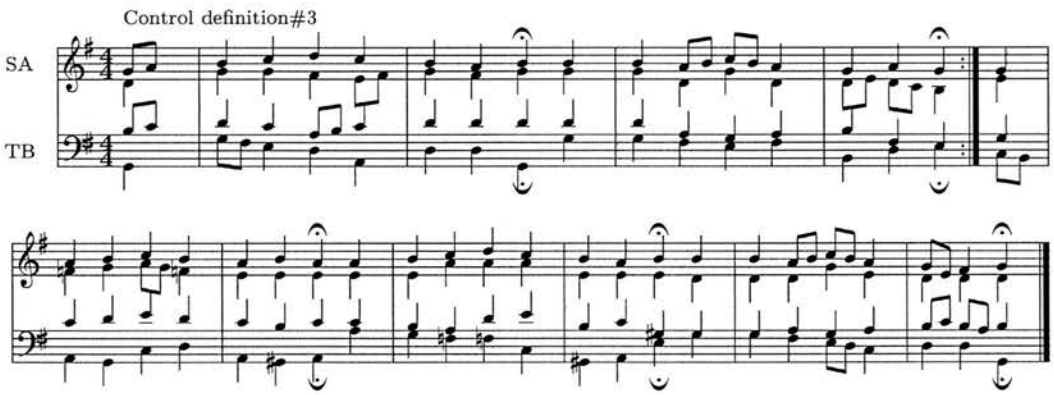
B.1.11 R080 O Haupt voll Blut und Wunden



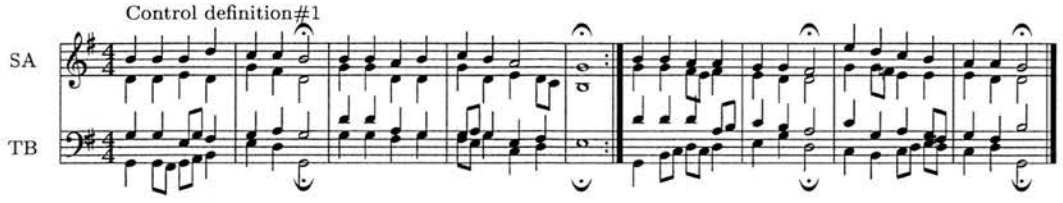
B.1.12 R088 Helft mir Gotts Gute preisen



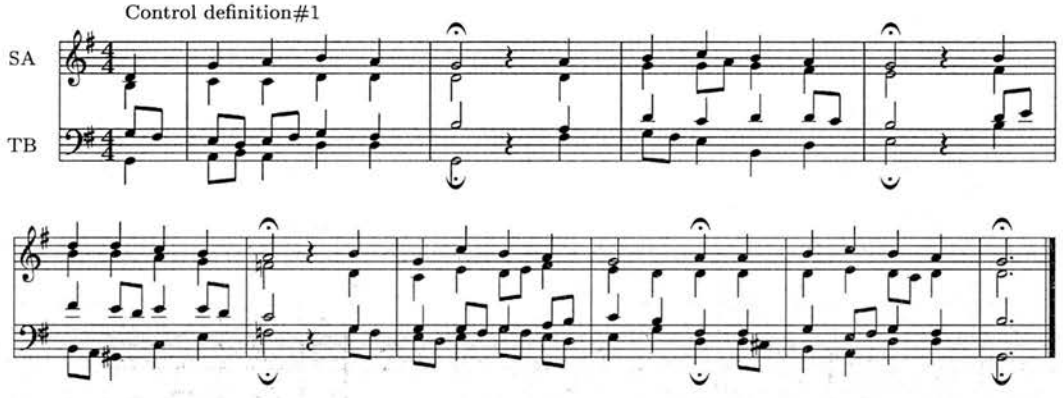
B.1.13 R125 Allein Gott in der Hoh sei Ehr



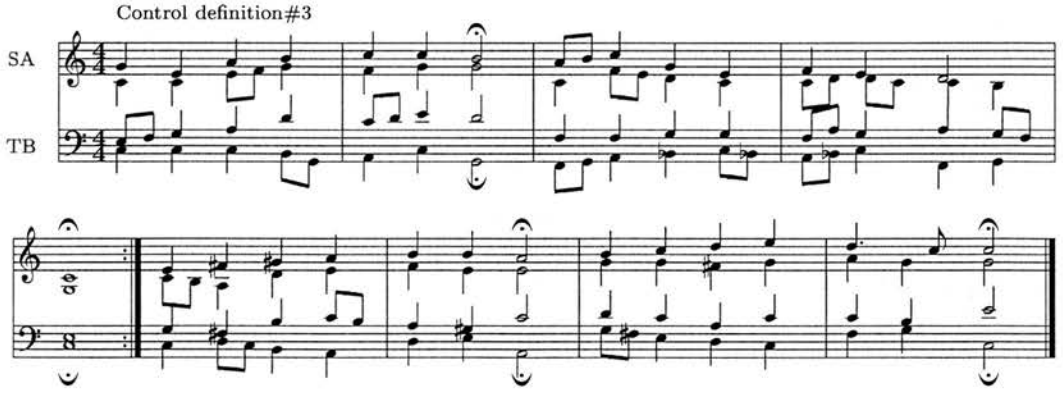
B.1.14 R151 Meinen Jesum lab ich nicht, Jesus



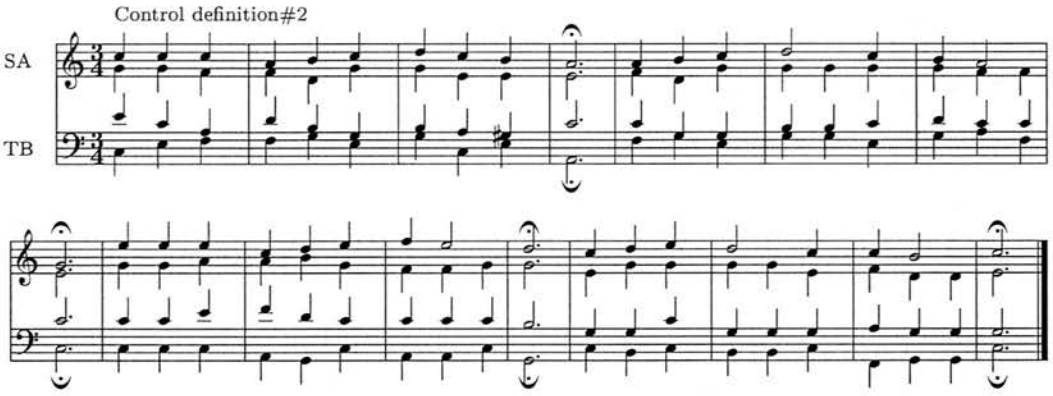
B.1.15 R159 Als der gutige Gott



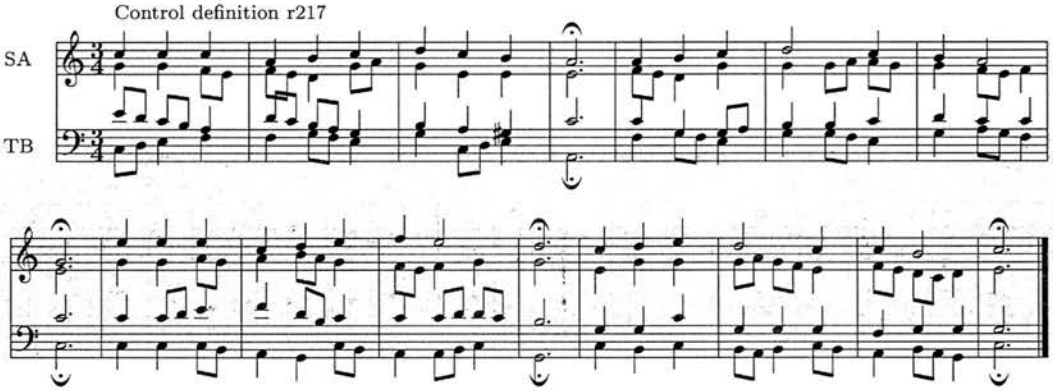
B.1.16 R175 Jesus, meine Zuversicht



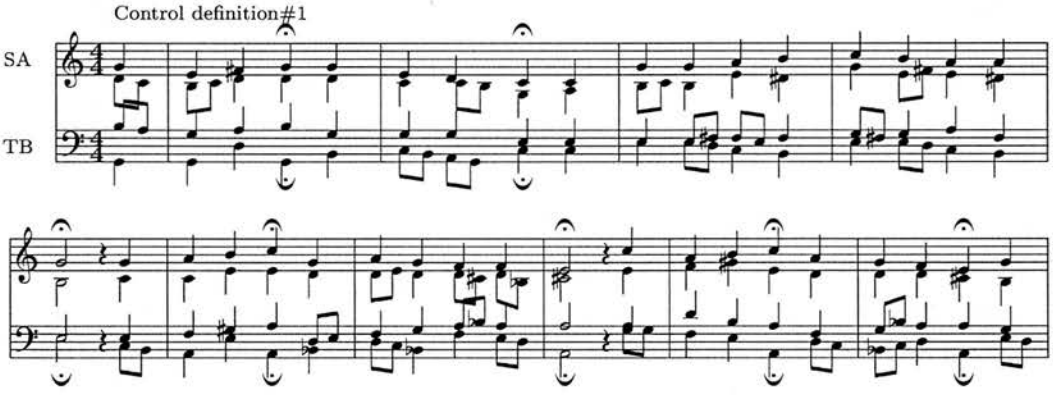
B.1.17 R217 Ach Gott, wie manches Herzeleid

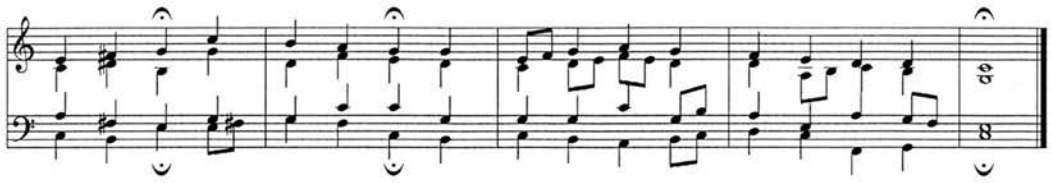


B.1.17.1 R217 Ach Gott, wie manches Herzeleid

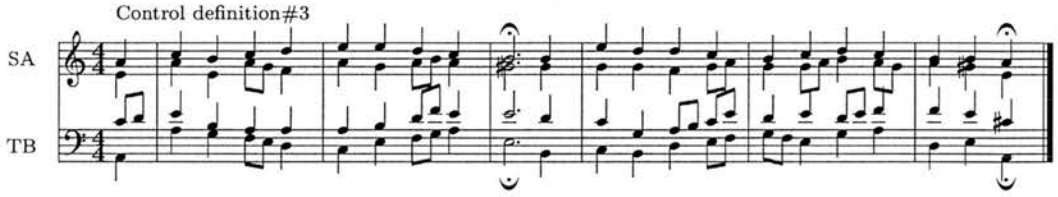


B.1.18 R223 Ich dank dir, Gott, fur all Wohltat

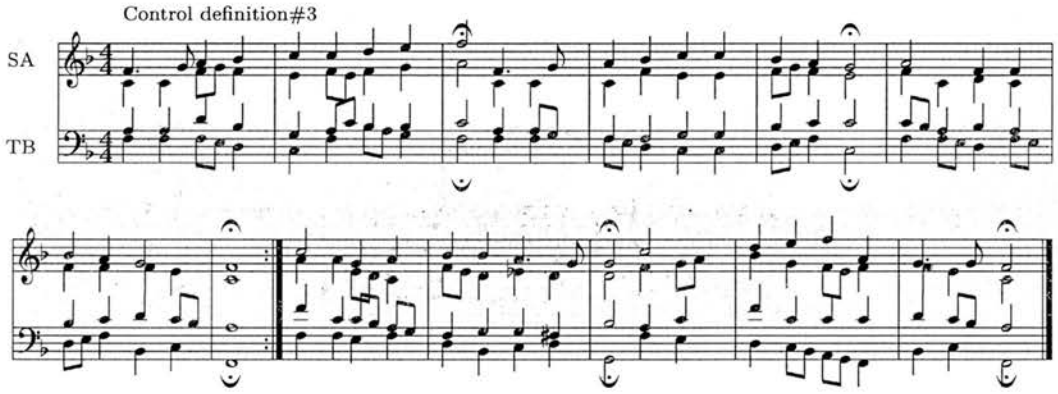




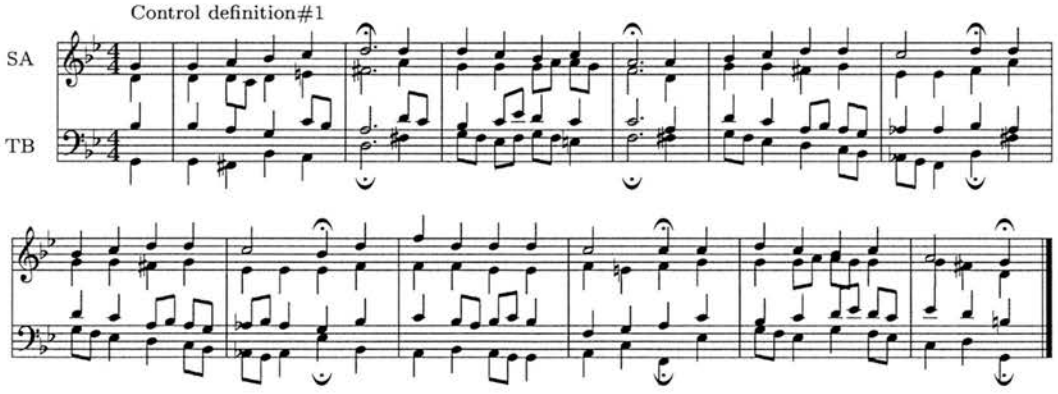
B.1.19 R228 Danket dem Herren, denn er ist sehr freundlich



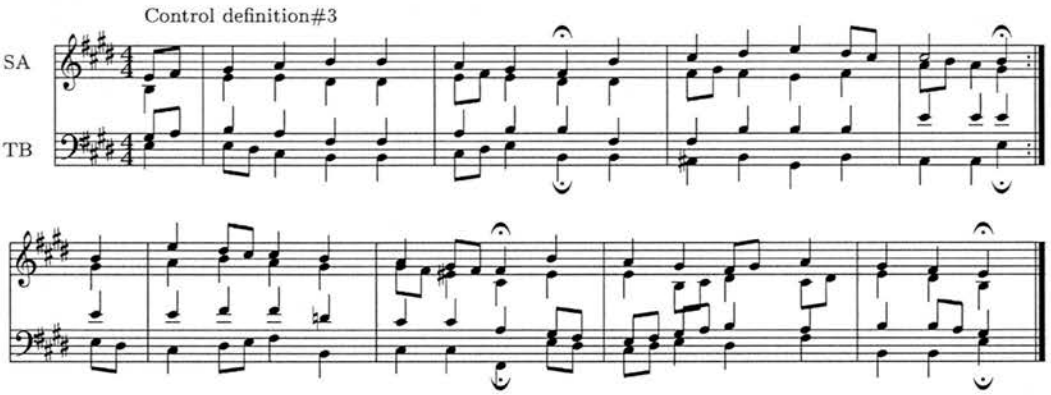
B.1.20 R274 O Ewigkeit, du Donnerwort



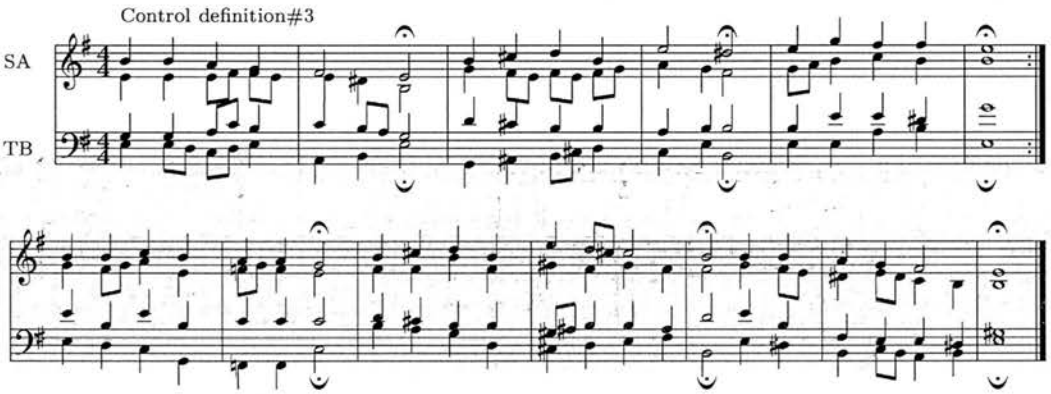
B.1.21 R304 Auf meinen lieben Gott



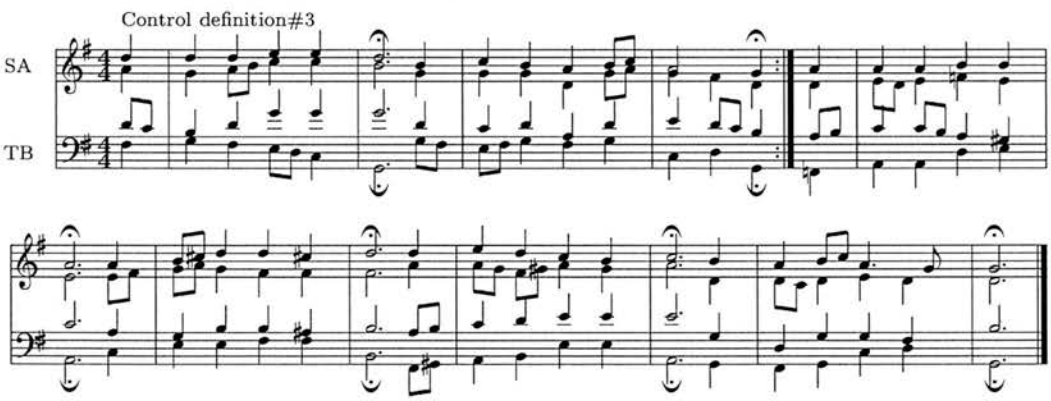
B.1.22 R310 Machs mit mir, Gott, nach deiner Gut



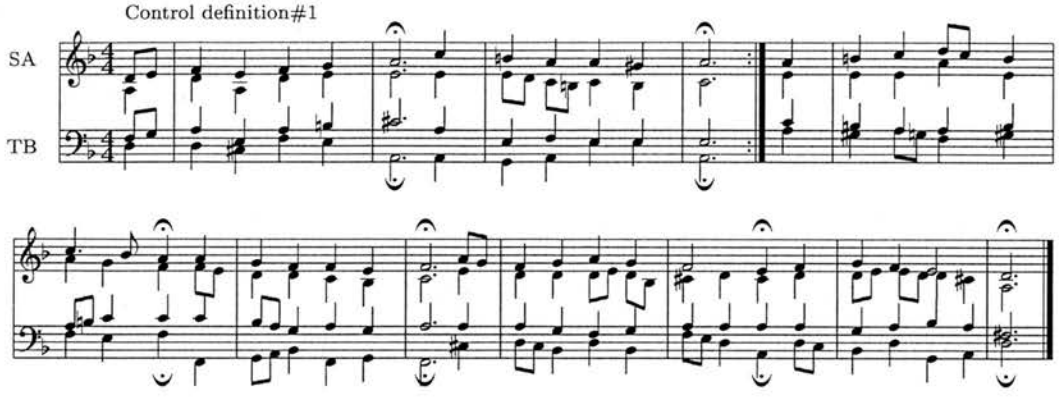
B.1.23 R324 Jesu, meine Freude



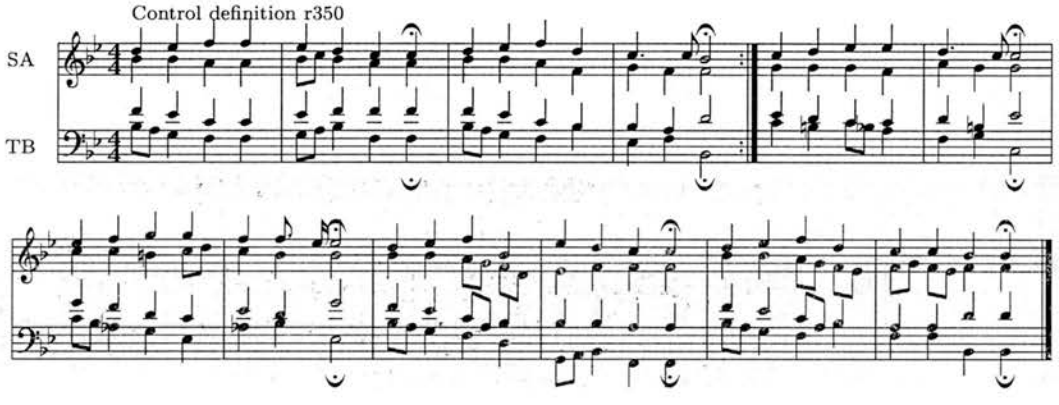
B.1.24 R330 Nun danket alle Gott



B.1.25 R340 Befiehl du deine Wege



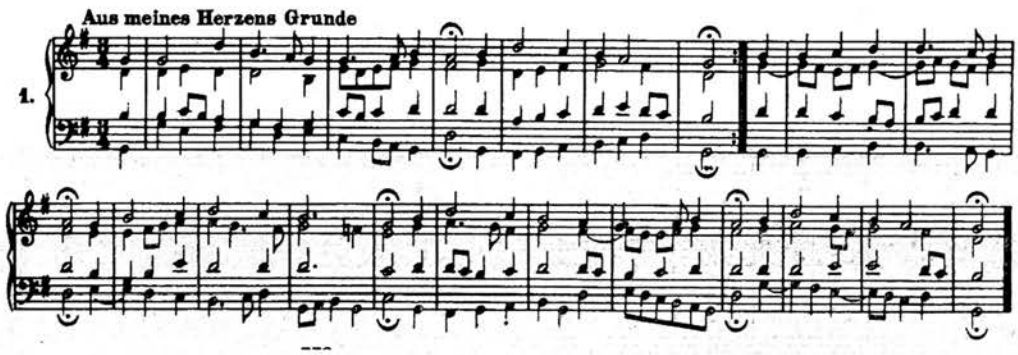
B.1.26 R350 Jesu, meiner Seelen Wonne



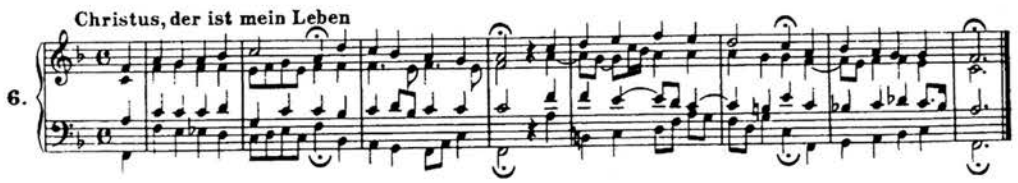
B.2 Bach's Original Harmonisations

The following harmonisations are scanned from the Riemenschneider [1941] '371 Harmonised Chorales and 69 Chorale Melodies with figured bass'. They are included here for readers' convenience.

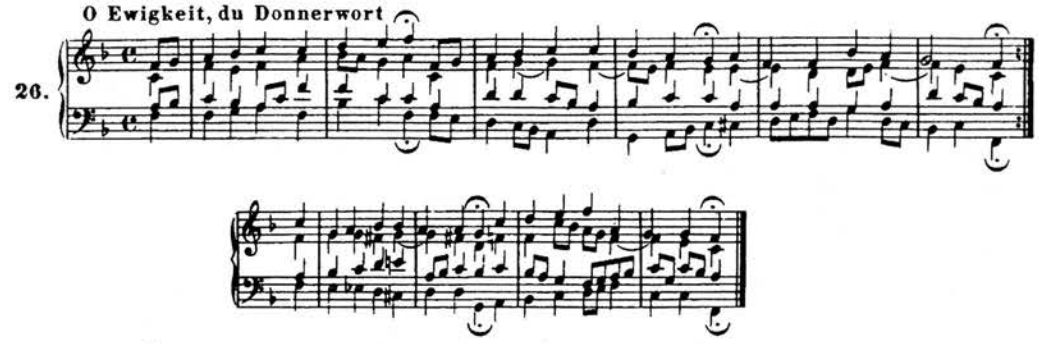
R001



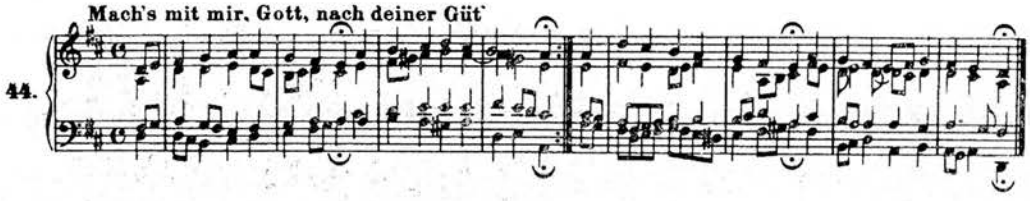
R006



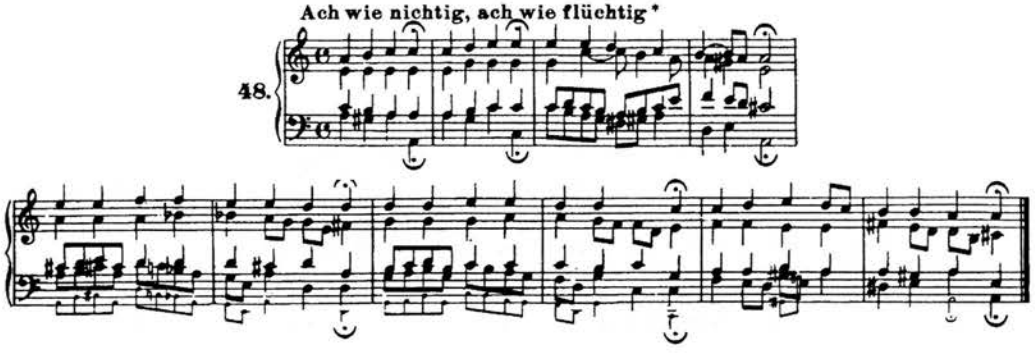
R026



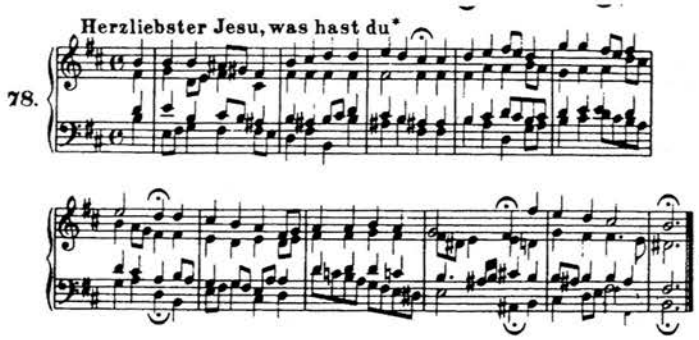
R044



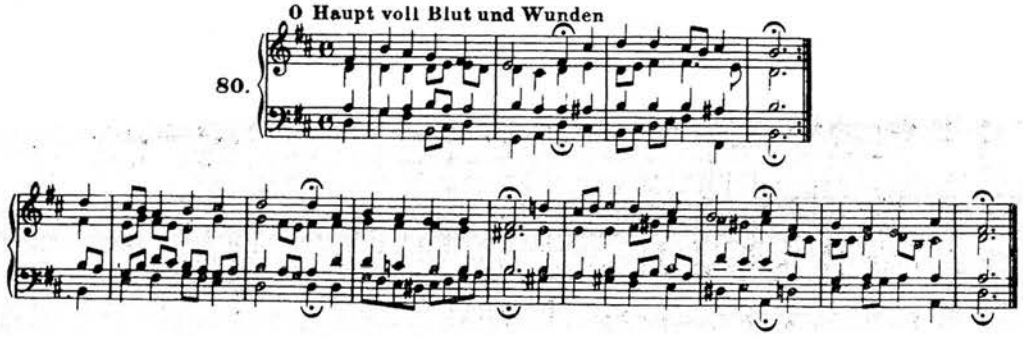
R048



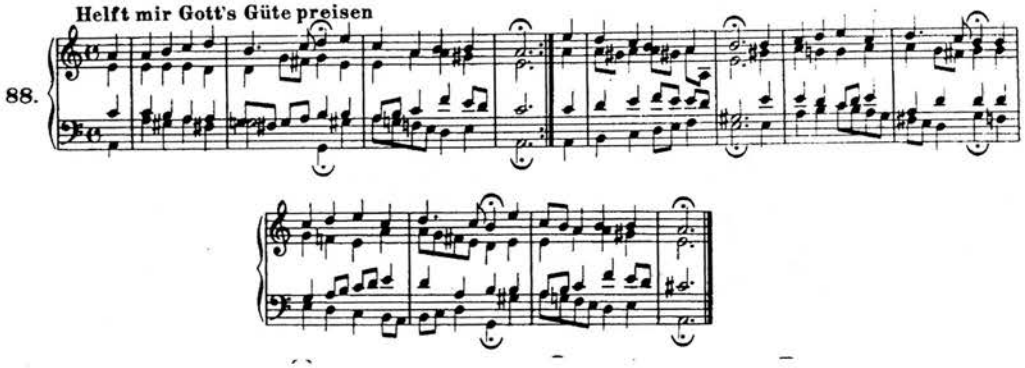
R078



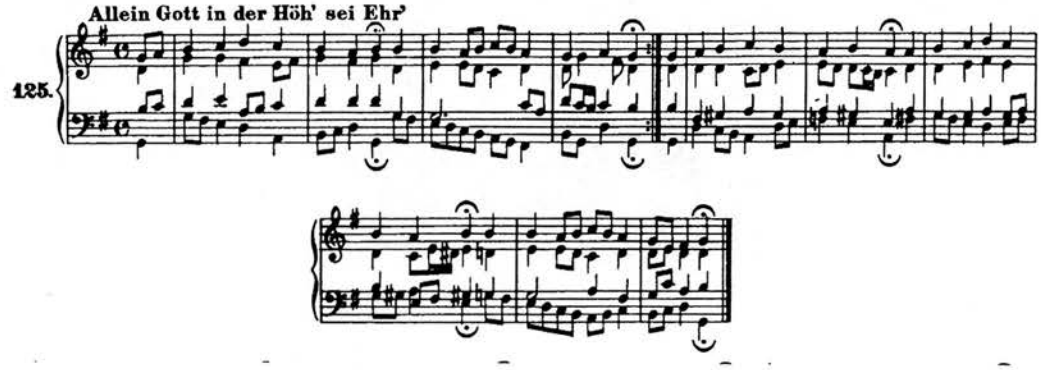
R080



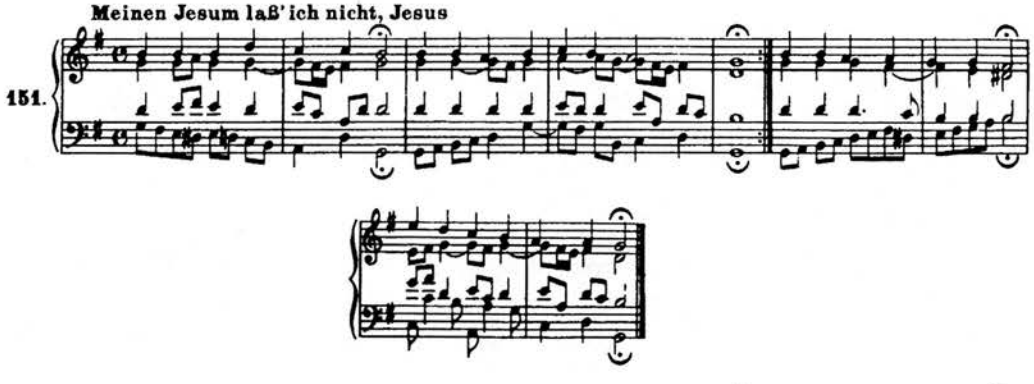
R088



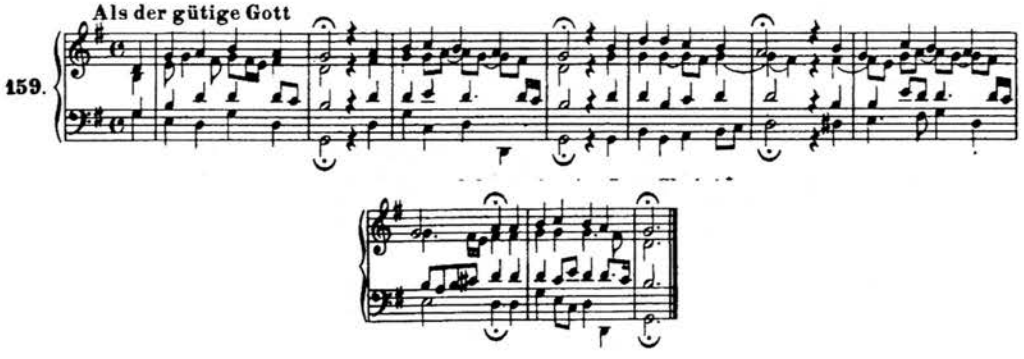
R125


Allein Gott in der Höh' sei Ehr'
125.

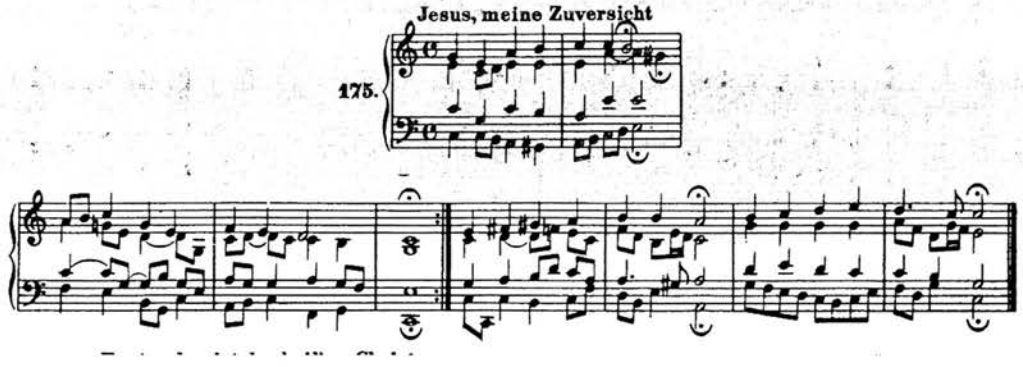
R151


Meinen Jesum laß' ich nicht, Jesus
151.

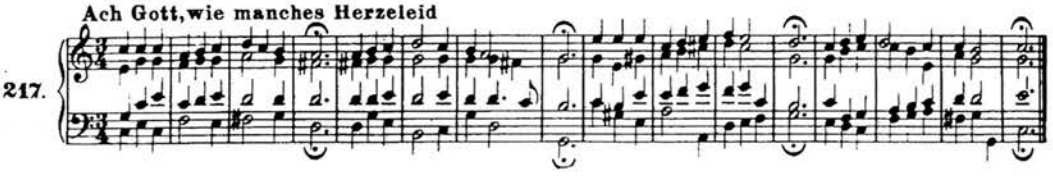
R159



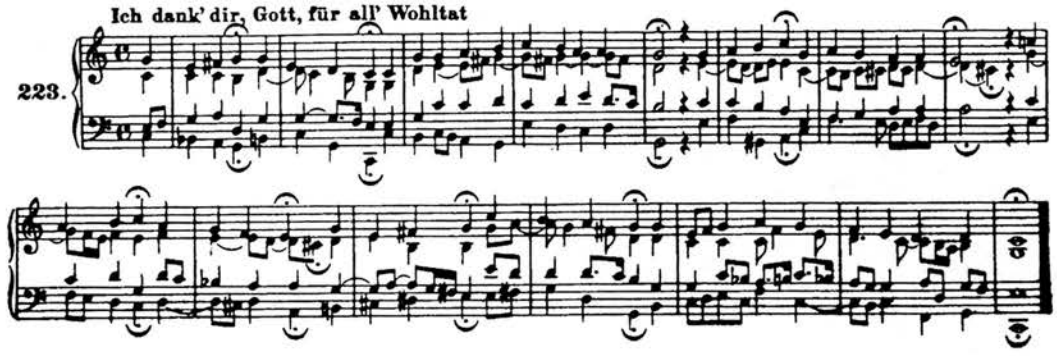
R175



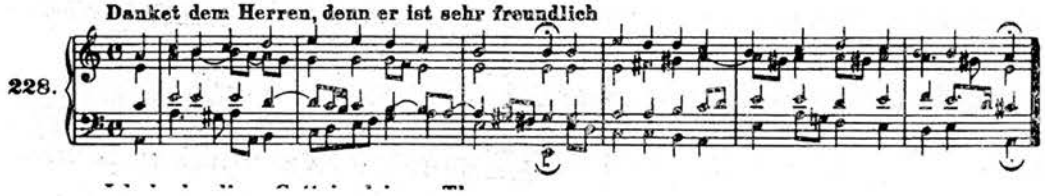
R217



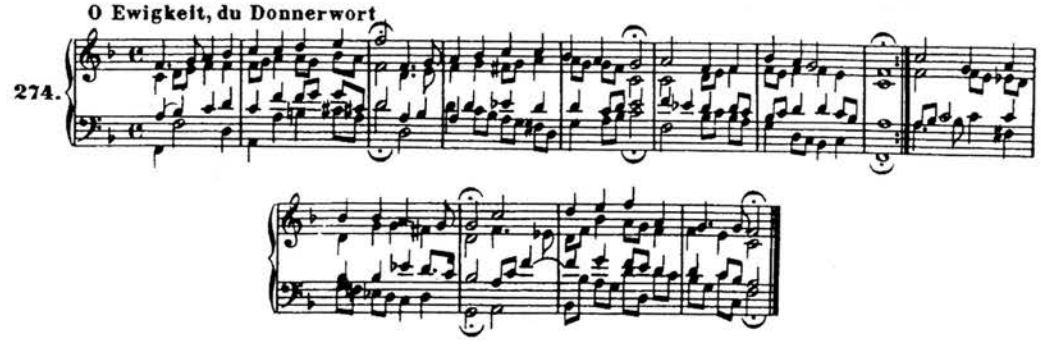
R223



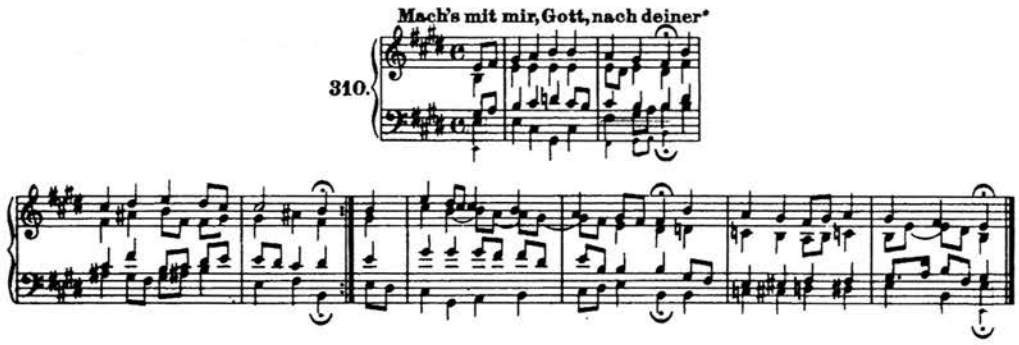
R228



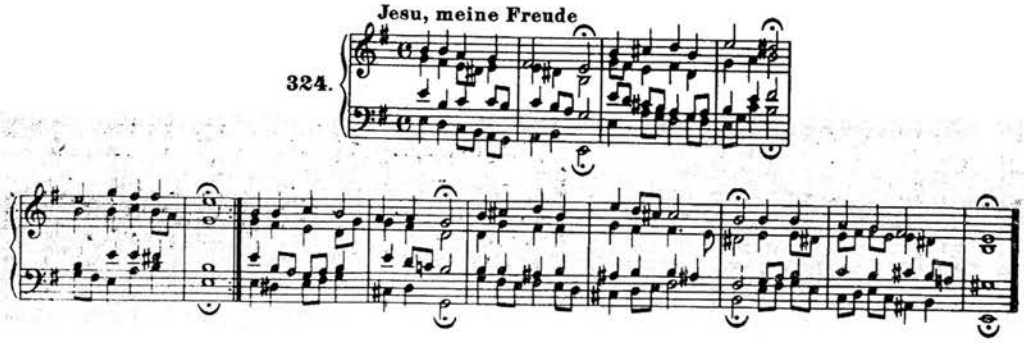
R274



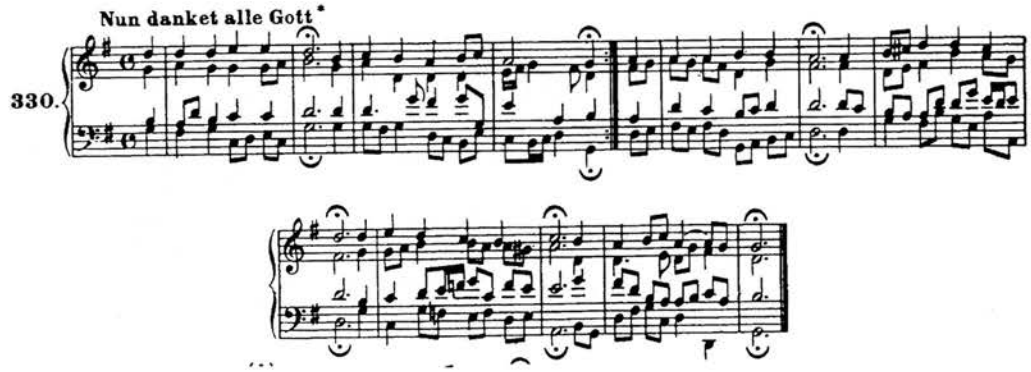
R310



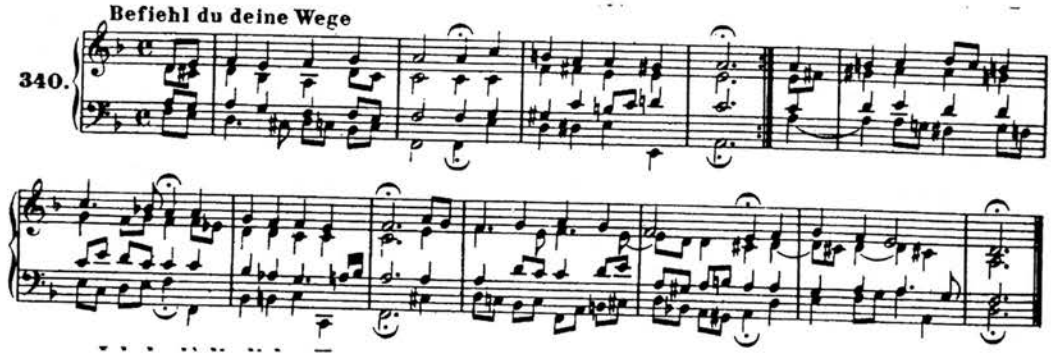
R324



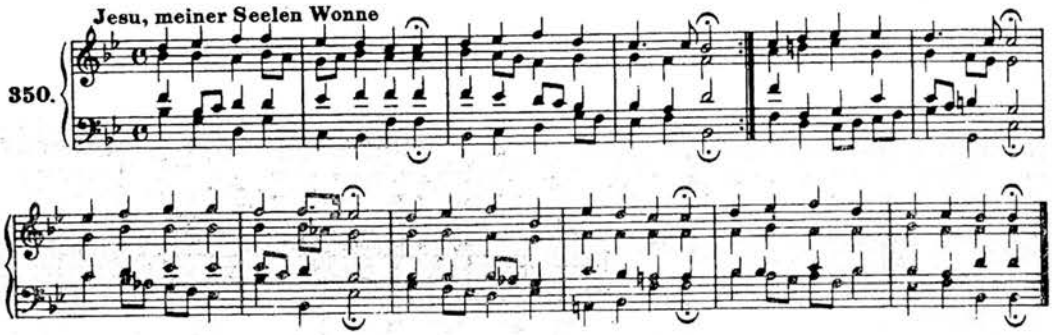
R330



R340



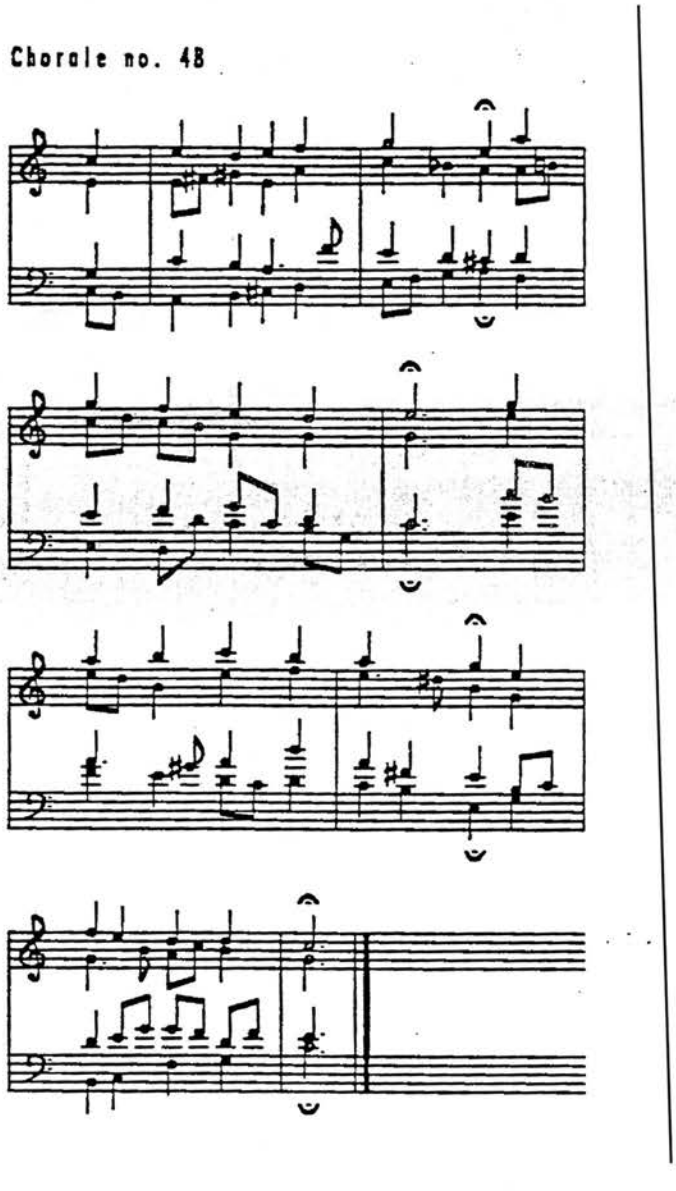
R350



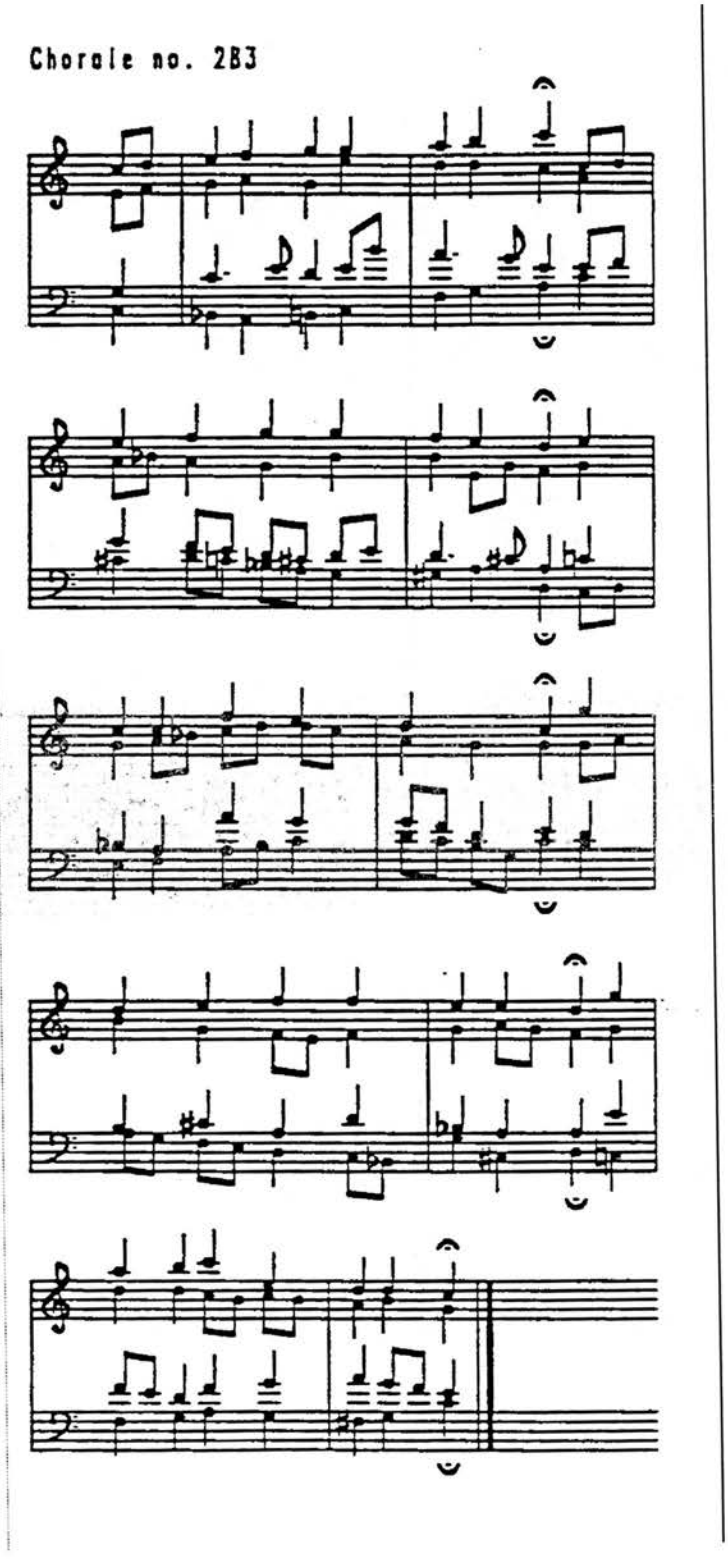
B.3 Examples from the CHORAL system

The following harmonisations are scanned from the 'Report on the *CHORAL* Project' [Ebcioglu, 1993]. All the examples below are transposed to the key of C major (for a major key chorale) and transposed to the key of A minor (for a minor key chorale). The examples are included here to show a flavour of the output from the *CHORAL* system.

R006

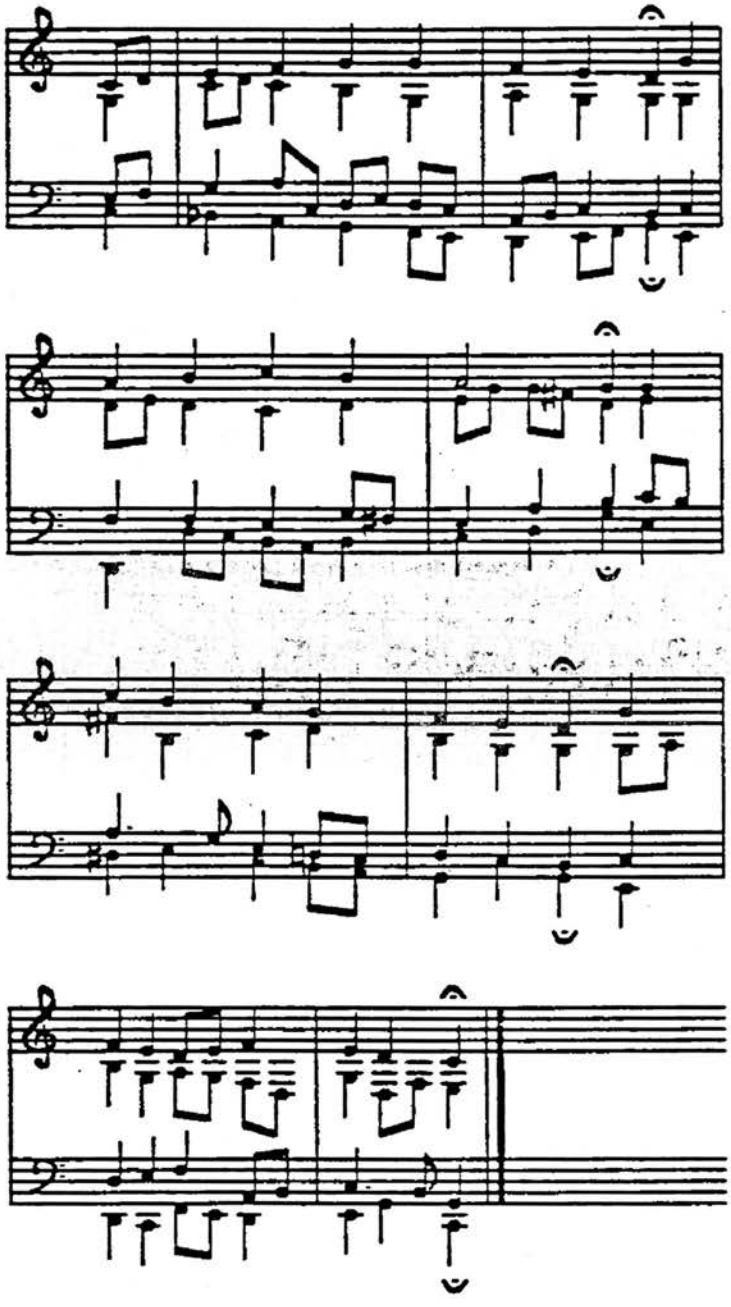


R026

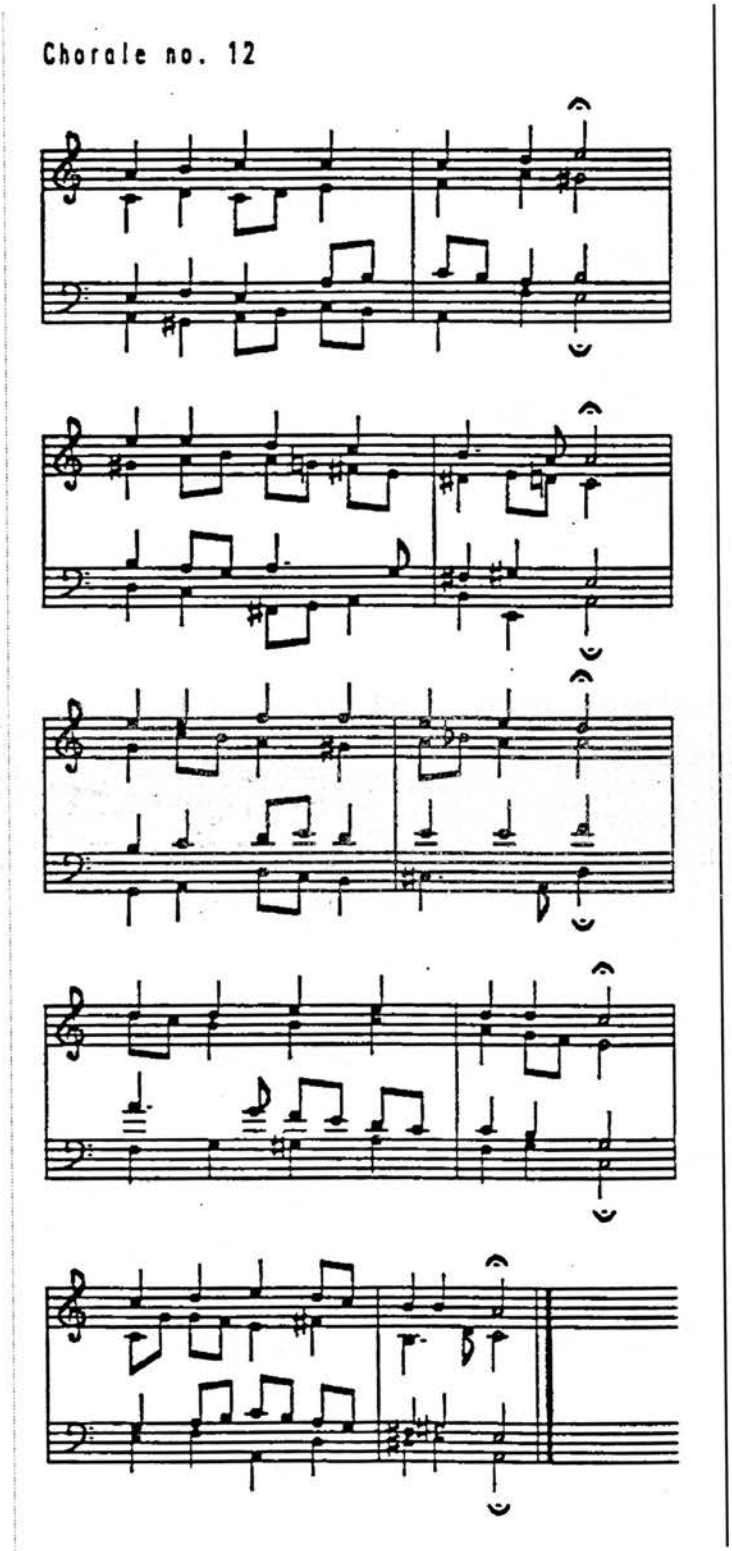


R044

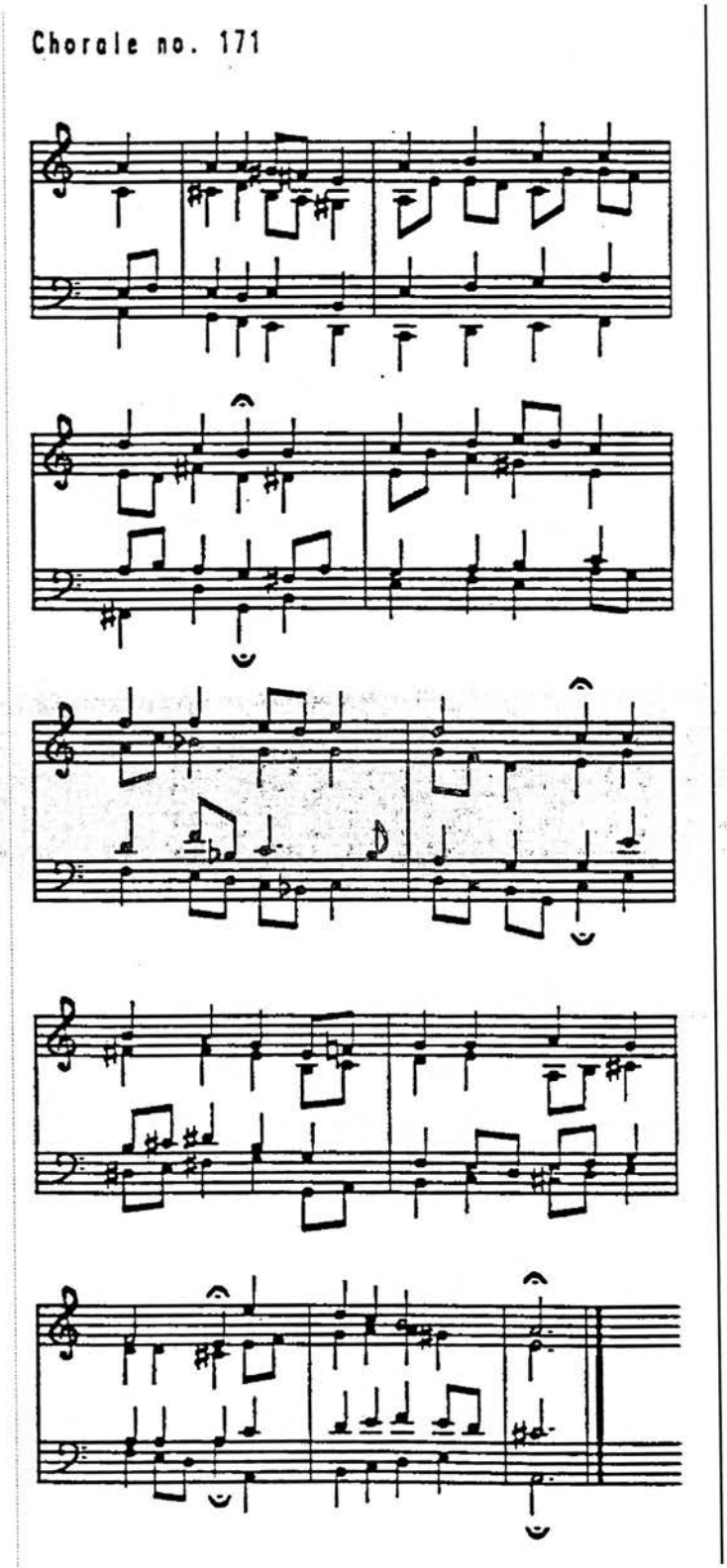
Chorale no. 241



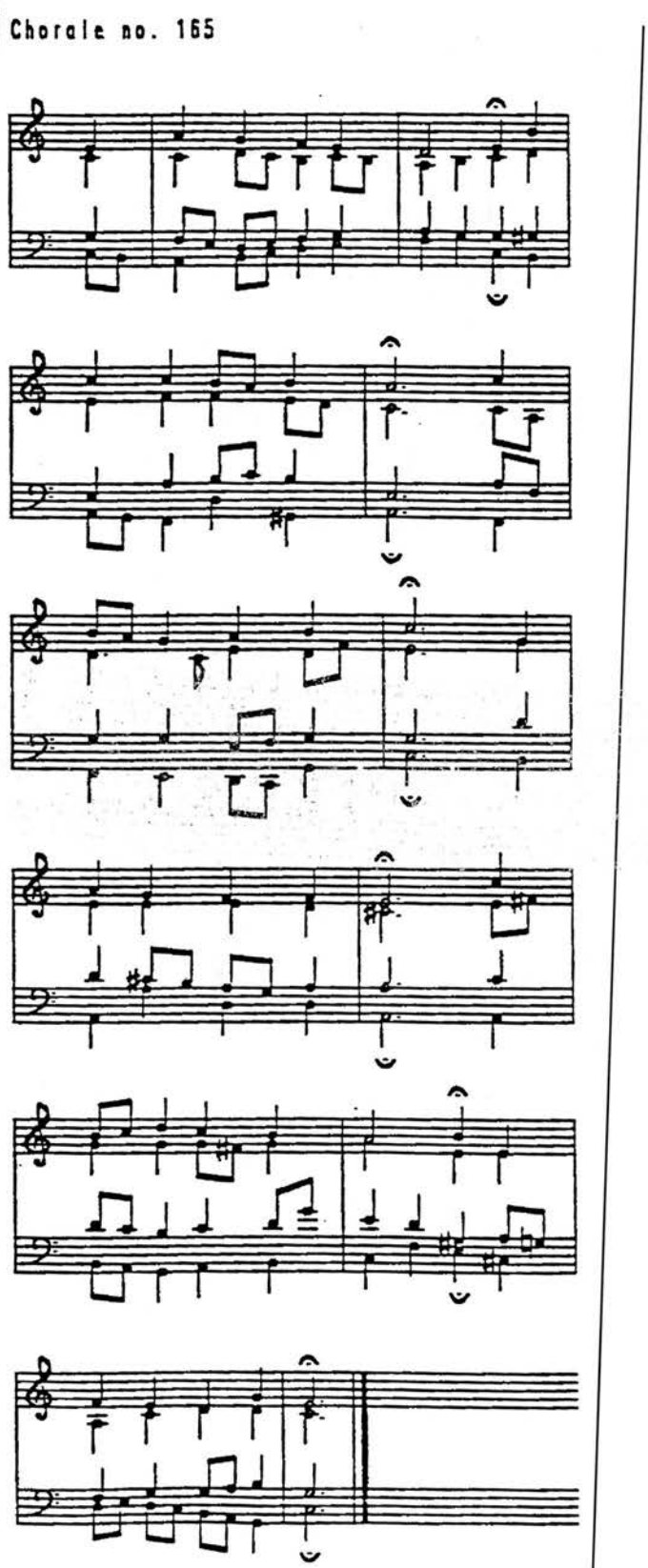
R048



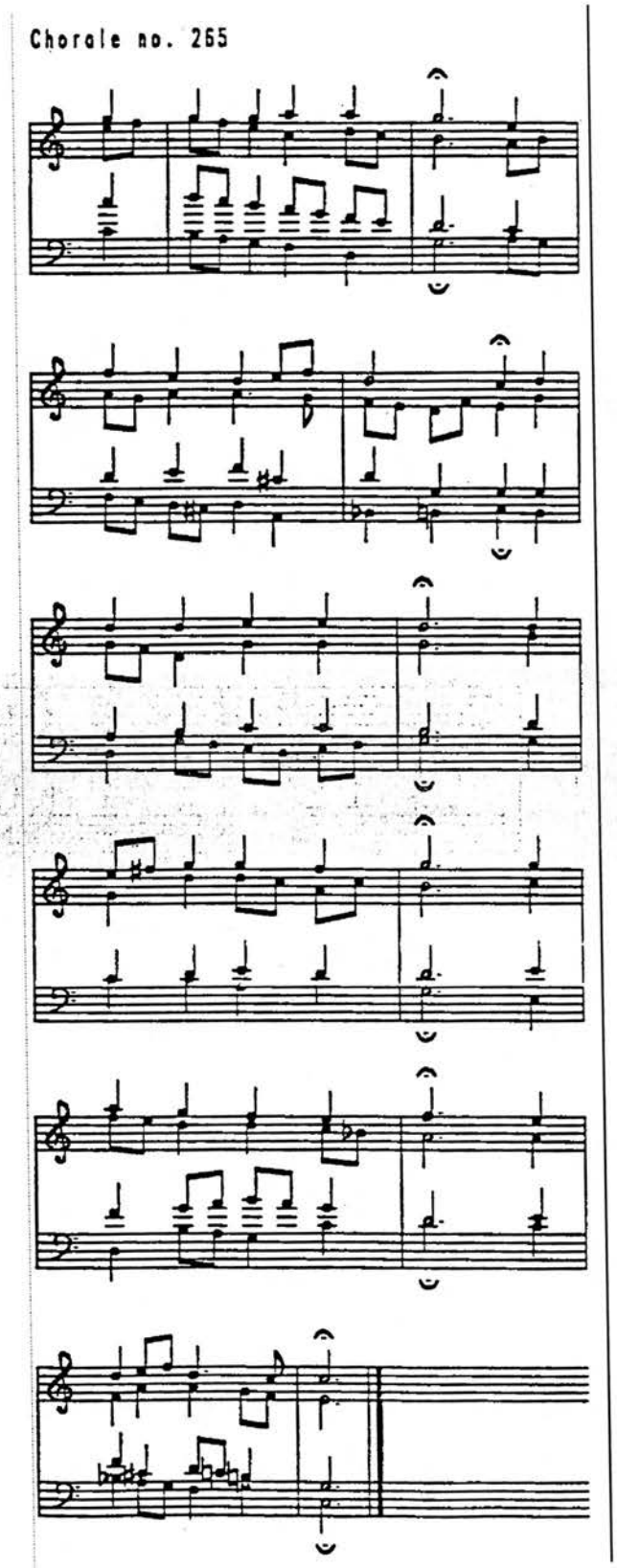
R078



R080



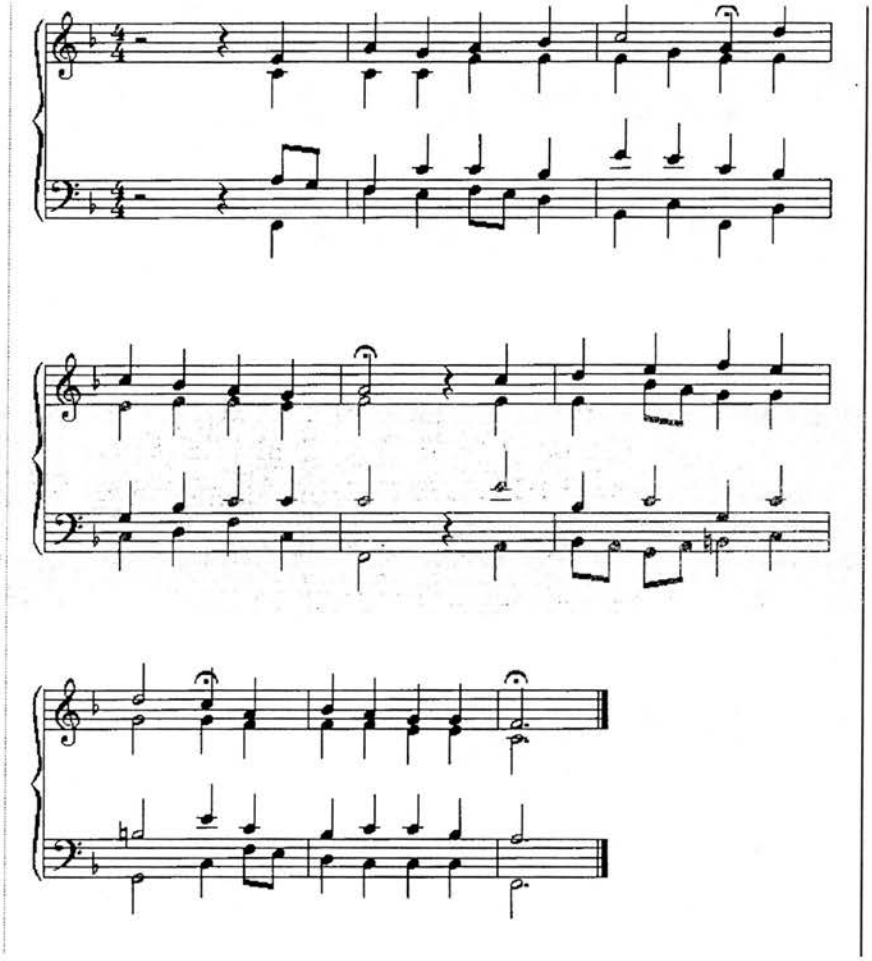
R330



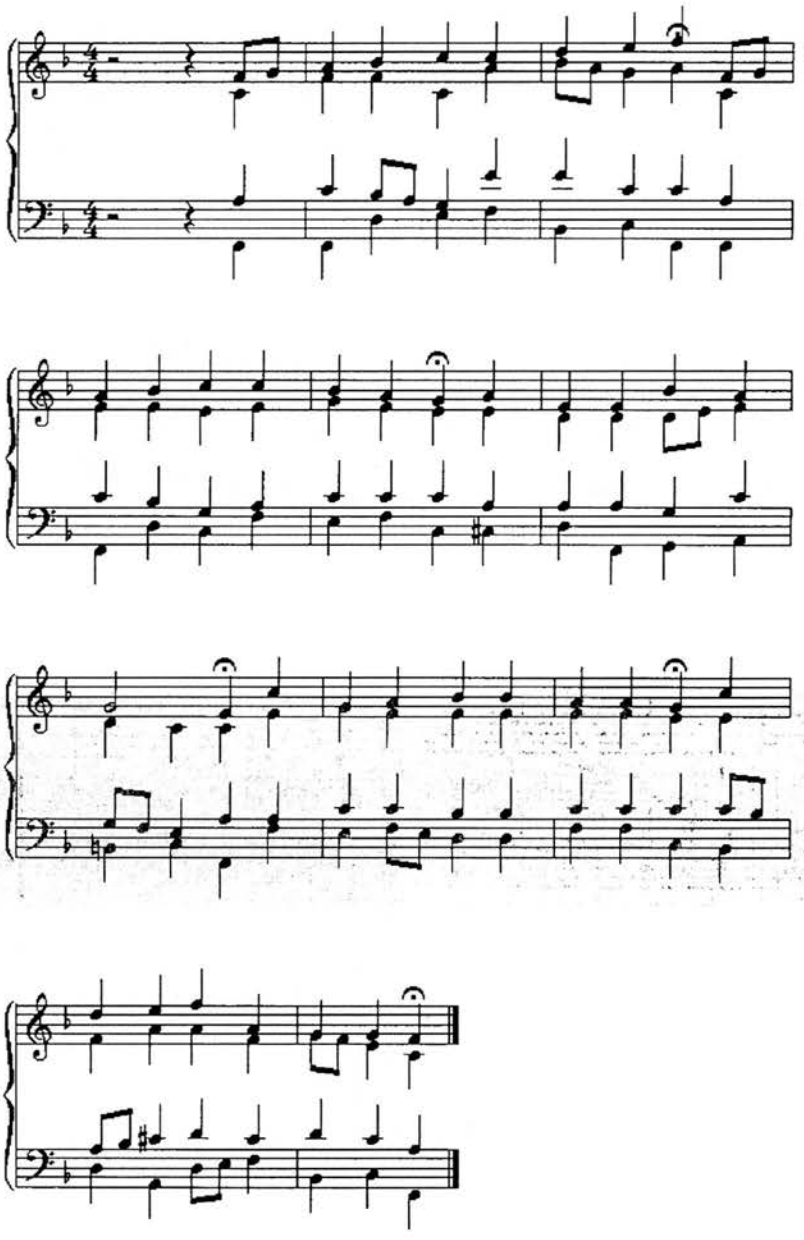
B.4 Examples from the HARMONET system

The following harmonisations are scanned from outputs produced from the *HARMONET* system. The examples are obtained from http://i11www.ira.uke.de/ musik/Folien/ in the period between October-December 2000. The examples are included here to show a flavour of the output from the *HARMONET* system.

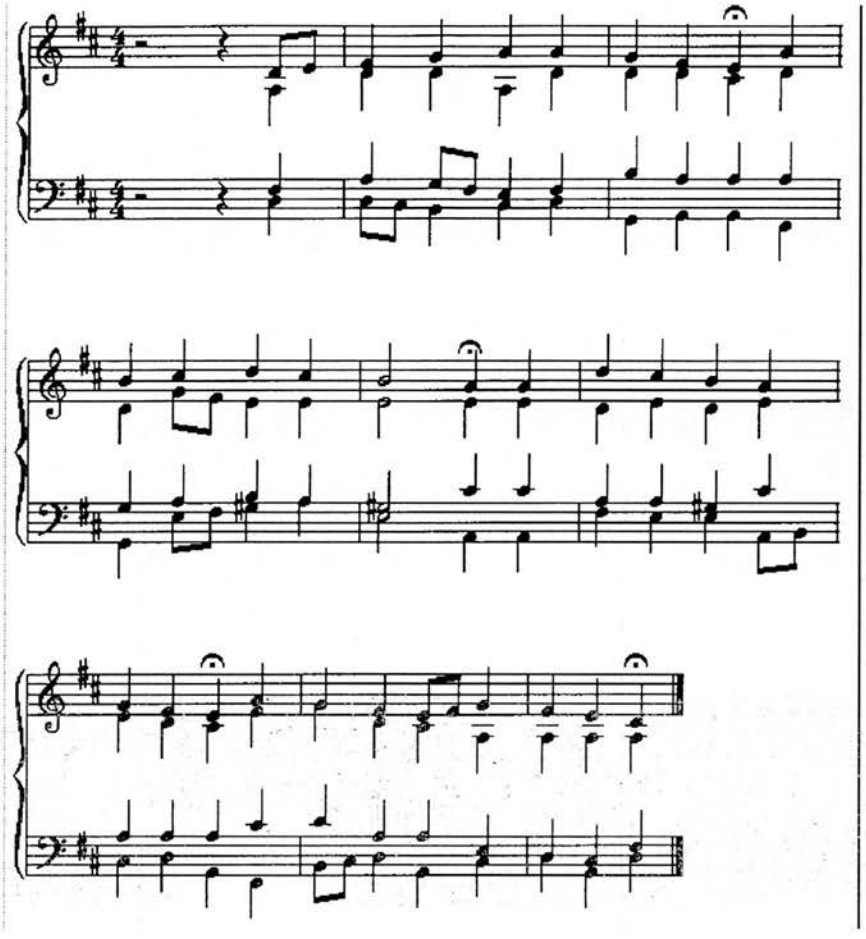
R006



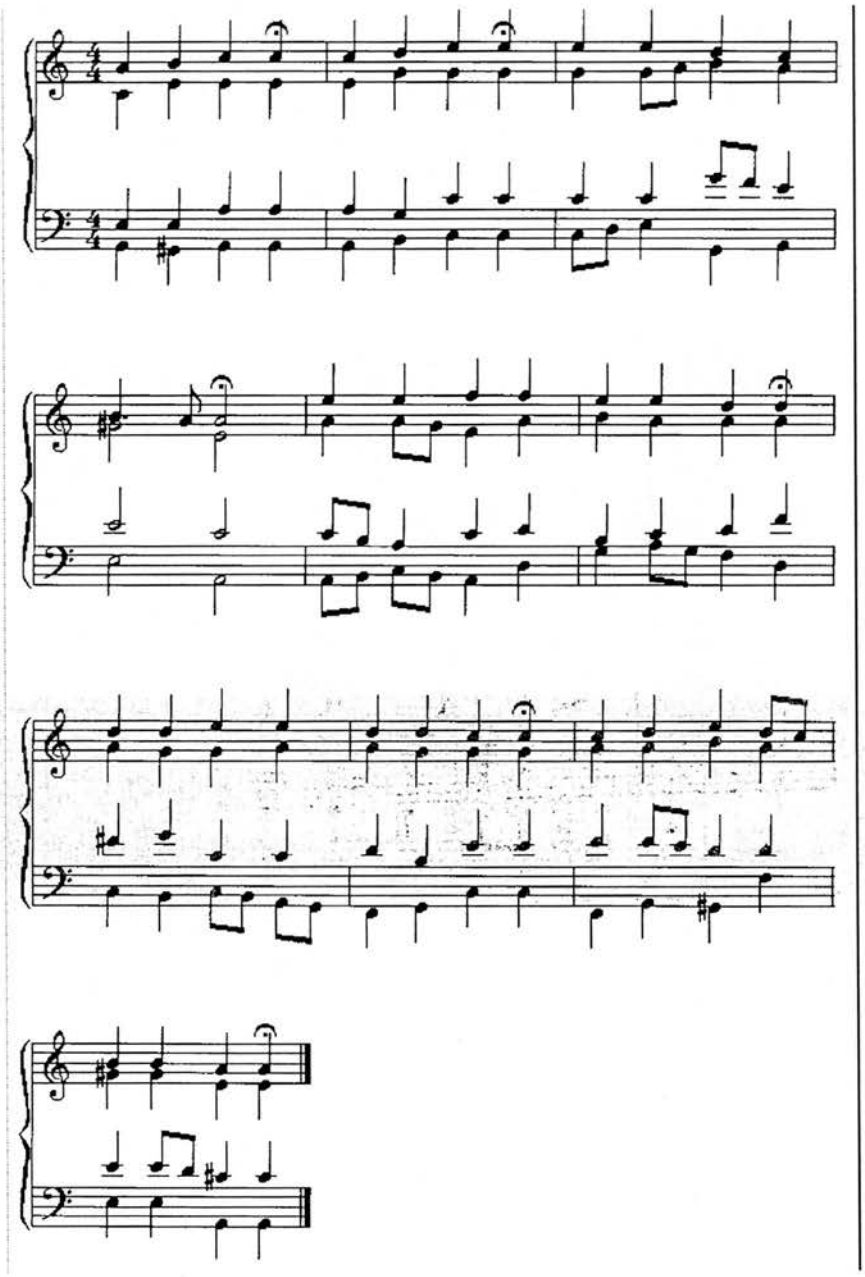
R026



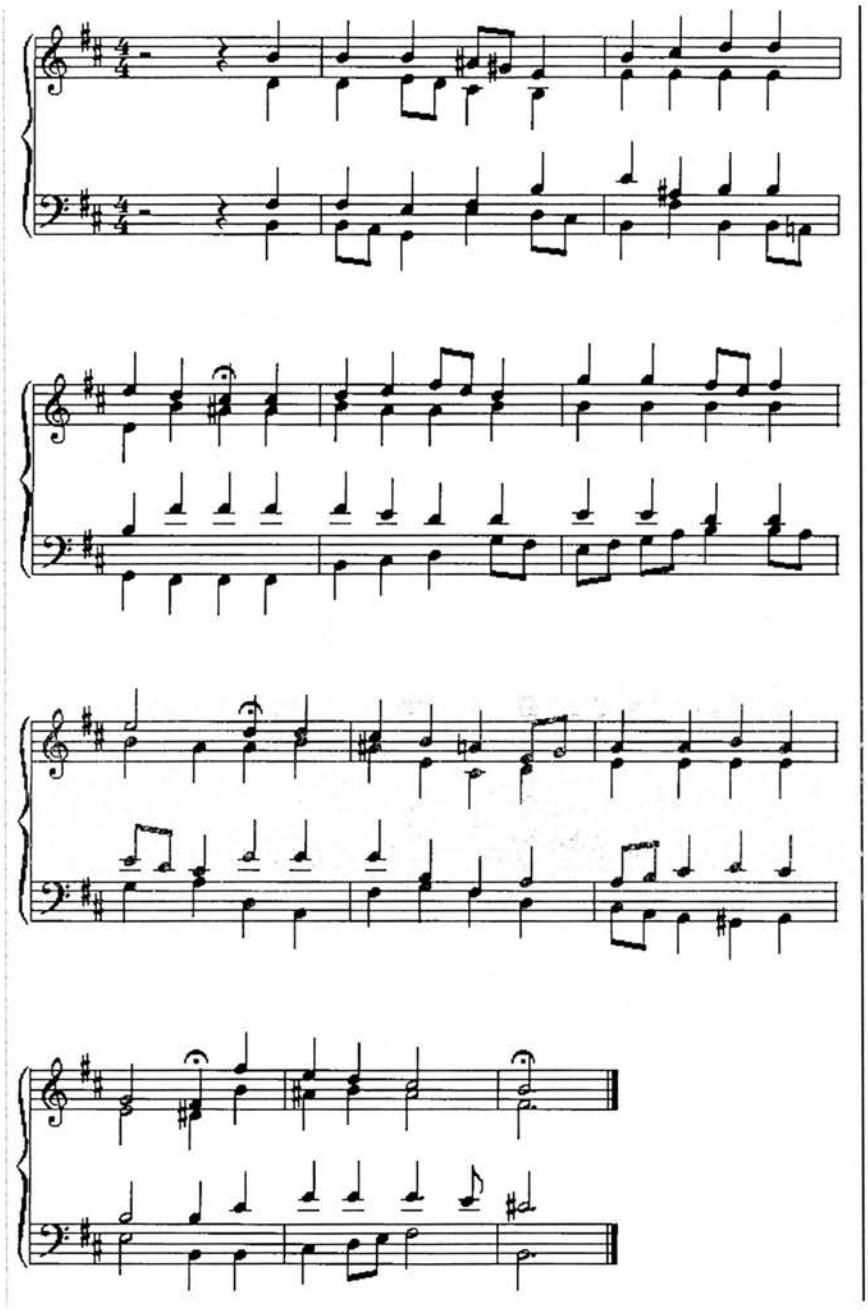
R044



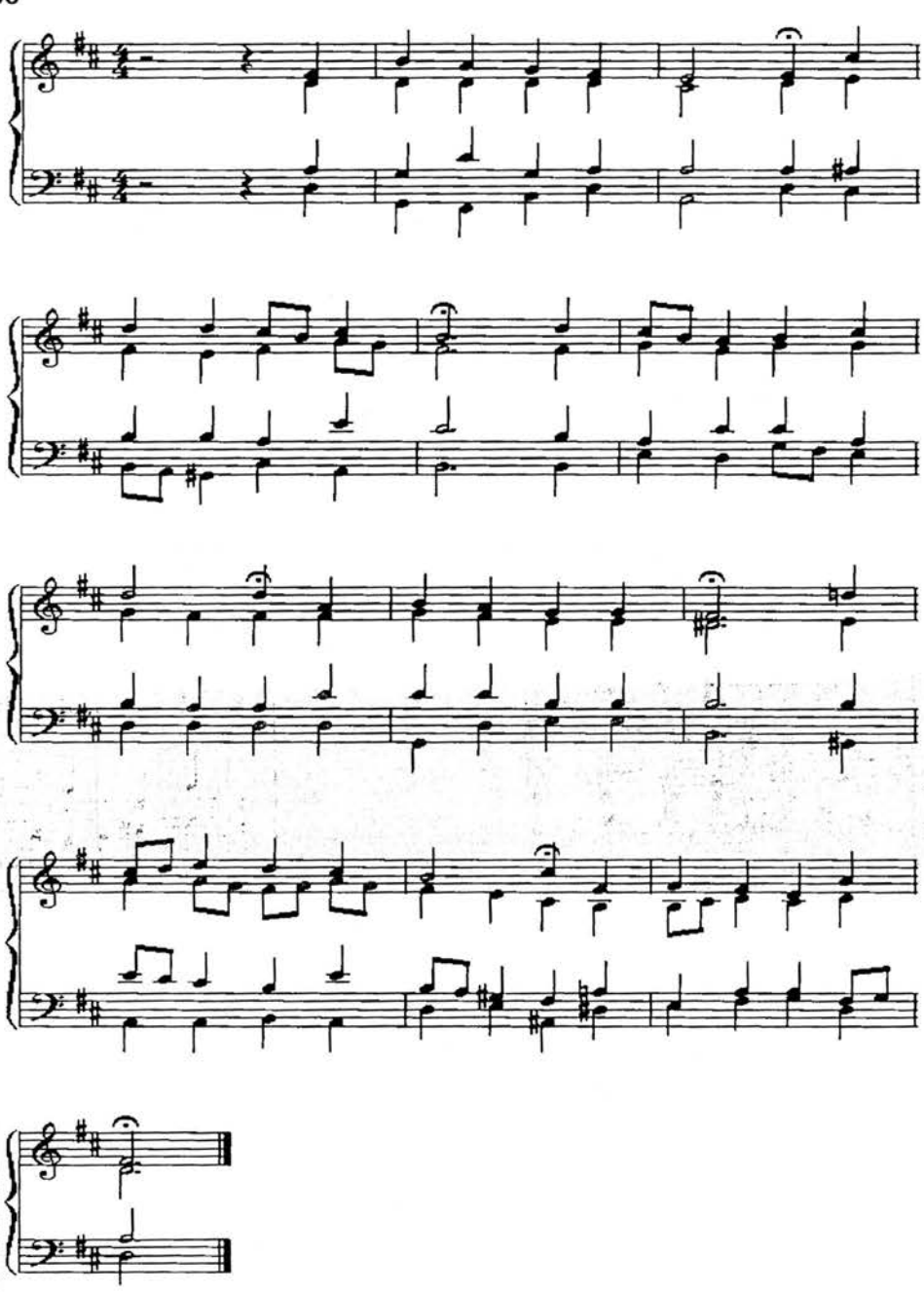
R048



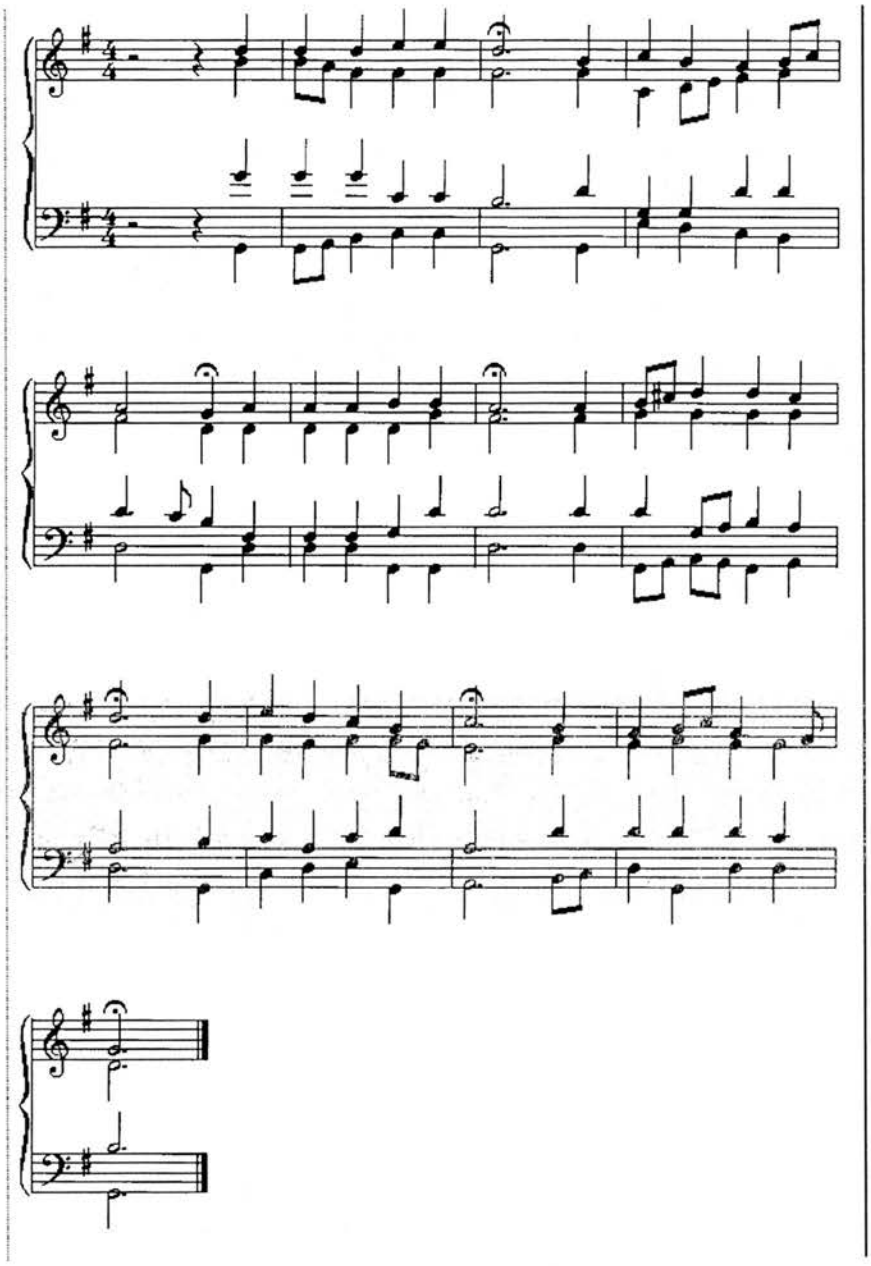
R078



R080



R330



Appendix C

CHORAL

C.1 Background

Ebcioglu [1987] attempts to imitate a human expert in harmonising four part chorales in the style of J.S Bach. His CHORAL system which harmonises a given melody in the style of Bach's chorales is said to have a competence approaching that of a talented student of music who has studied Bach's chorales. The Choral system is implemented with *BSL* (Backtracking Specification Language) with three main components in its program structure:

- Generate section: This uses condition–action pairs which have the informal meaning: IF conditions are true about partial solutions, THEN a new element as described by the actions can be added to the partial solutions.
- Test section: This contain constraints which can reject certain assignments of the partial solutions.
- Recommendations section: This is a heuristic section which asserts what is desirable about the partial solution.

Ebcioglu realises that different knowledge representations suit different tasks. He captures musical knowledge in different *viewpoints*:

- A chord skeleton view: The chord skeleton view generates a chord skeleton for a chorales. The knowledge of key, voicing in a chord, chord types, chord transitions, cadence types is captured in this view.
- A fill-in view: The fill in view fills in the available chords with notes to create

chorales. This view allows the fill-in of suspensions, passing notes and neighbour notes for each voice.

- A melodic-string view: The melodic-string view observes the sequence of individual notes in each voice from the melodic point of view. The voice progression constraints are placed in this view.
- A merged melodic-string view: This view is similar to the melodic-string view, except that repeated pitches are merged into a single note.
- A time-slice view: observe the chorales as a sequence of vertical time-slices (of quaver note duration). The general harmonic constraints and preferences, rhythmic constraints and preferences are specified in this view.
- A schenker parser: The parser executes parser steps on the soprano line using the Schenkerian concept.

The CHORAL system controls the search through the search space with an intelligent backtracking system. The intelligent backtracking allows the backtracking process to backtrack directly to the problem step, thus avoiding backtracking to all steps as in an ordinary chronological backtracking system. The system works on the input melody from left to right in a cycle of the *chord skeleton view*, the *fill in process* (which includes the fill in view, the melodic string view, the merged melodic string view and the time slice view in one process) and the *Schenkerian analysis view*.

There are discrepancies in harmonisations between the Terry collection and the Riemenschneider collection in these chorales. The chorales used in the Ebcioglu's CHORAL report are based on the Terry collection; from hereafter, we will refer to chorales in the Terry collection with letter 'T' and the Riemenschneider collection with the letter 'R'.

C.2 The Knowledge in Ebcioglu's CHORAL Work

This appendix attempts to summarise the knowledge content in the CHORAL system in a declarative form. However, in some situations, where the declarative concepts may be ambiguous, the imperative format may be chosen for a more concise way of expressing things. In our opinion, reading the knowledge content in the CHORAL report is not an easy task. Since we have been looking at the knowledge content in the CHORAL system and have reused some of the knowledge content, we hope this could offer some useful information for researchers in this area.

The knowledge content of the Chord skeleton view and the Fill-in process in the CHORAL system is grouped into four main areas and elaborated [see Ebcioglu, 1987, appendix B]. The number in brackets used hereafter refers to the relevant section in the appendix B in Ebcioglu's original CHORAL report. The four main knowledge groupings are:

- The basic building blocks of the knowledge representation in the relevant views.
- The generation of knowledge from the above building blocks which determines the search space the system can address.
- The constraint part which prunes out unwanted parts of the search space.
- The heuristic part which assigns different preference to different branches of the search tree.

We summarise the knowledge content in the Chord skeleton view and the Fill-in process according to these groupings. Due to the nature of the knowledge in the CHORAL system which contains both the specific control knowledge and the domain knowledge of the system, the knowledge content which is related to the control process is omitted from this summary.

C.2.1 Basic building blocks in the Chord skeleton view (1.1.1)

The chord skeleton view sees a chorale as a rhythmless chord sequence. The basic building blocks of the view are summarised below:

- Knowledge of pitches: Pitch names, Accidentals and Octave registers are used to denote the pitch attributes of sound.
- Knowledge of chords: Triads, Seventh chords, Inversion types, Chord types, Chord functions, Root and Root accidental are used to categorise groups of pitches as chords.
- Knowledge of keys: CHORAL operates under the following keys: C major, A minor, G major, E minor, F major and D minor.
- Others: There are other basic building blocks associated with the control in the CHORAL system: chord sequence number, cliche pointer, force suspension, max note, min note, phrase counter, current time, last high corner, etc. We will discuss these items only when necessary.

Notations used in CHORAL				Notations used in the thesis	
Symbol	Major	Minor	Remarks	Major mode	Minor mode
stI	x	x		I	i
stI6_4	x	x	see a	I_c	i_c
stII	x	x		ii, ii_7	ii^o,ii^o_7
stIIp	x		see b	ii, ii_7	
stIIu		x	see c		ii
stIII	x			iii	III^+
stIIId		x	see d		III
stIV	x	x		IV	iv, iv_7
stIVp	x		see b	IV	
stIV7	x			IV_7	
stIVu		x	see c		IV, IV_7
stV	x	x		V, V_7	V, V_7
stVd		x	see d		v
stVI	x	x		vi, vi_7	VI
stVII	x	x		vii^o	vii^o, vii^o_7
stVIId		x	see d, e		♮VII

a: Cadential $\frac{6}{4}$ chord

b: A passing chord between two dominant chords

c: Chords built from ascending a melodic minor scale

d: Chords built from descending a melodic minor scale

e: Note that the root accidental depends on the key signatures

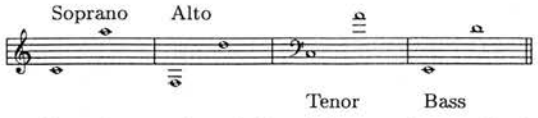
C.2.2 Generation section in the Chord skeleton view (1.1.2—1.1.5)

C.2.2.1 Generation of utility attributes (1.1.2)

The program starts with initial default values. All attributes of step n are computed from step $n-1$.

C.2.2.2 Generation of pitch and accidental of all voices(1.1.3)

- Voice range



- Pitches are selected to form a legal chord for major and minor mode (see Figure 3.2 for legal chord types). In the case of triads, one pitch will be doubled for a four voice chord. An incomplete chord is not allowed at the chord skeleton level.
- Voices in the chord are checked so that:

  -Each voice is in its own range

  -The distances between soprano–alto and alto–tenor are not more than an octave

  -The distance between bass–tenor is not more than an octave plus a major second

  -There is no crossover of voices
- The following alterations are possible: $\hat{f} \rightarrow \hat{f}\sharp$; $\hat{g} \rightarrow \hat{g}\sharp$; $\hat{c} \rightarrow \hat{c}\sharp$; $\hat{d} \rightarrow \hat{d}\sharp$; $\hat{b} \rightarrow \hat{b}\flat$. This is to allow modulation to related keys. Ebcioglu gives more elaborate constraints of chord generation by restricting certain combinations[1] (see [Ebcioglu, 1987, p 239]).

C.2.2.3 Generation of key and harmonic degree within the key(1.1.4)

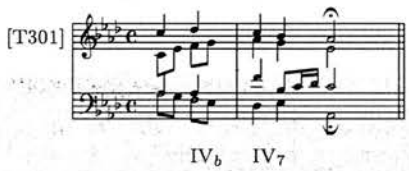
- The legal chord type and pitches are all possible inversions of chords in figure 3.2.
- Legal keys in the CHORAL system are C major, F major, G major A minor, D minor and E minor.
- The CHORAL will maintain the current key as long as there are no accidentals which disagree with the current key in the melody.
- If there is an accidental which is not in the current key, then the CHORAL changes key. The introduction of accidentals is conditioned by the accidental generation criteria mentioned earlier; or from the soprano itself (see [Ebcioglu, 1987, 1.1.3.3 p238]).
- The chorale must start with a tonic or a dominant chord.
- Summary of chord transition from left to right in a major mode:

  – no repeat of passing chord
  – no repeat of cadential $\frac{6}{4}$ chord (I_4^6)
  – no repeat of iii,vii^o
  – I $\rightarrow$ ii, ii$_7$, IV, vi, vi$_7$, V, V$_7$, vii^o
  – ii, ii$_7$ $\rightarrow$ V, V$_7$; passing ii $\rightarrow$ V, V$_7$

[1] We see these as the constraints to allow all possible combinations of all legal chords in all legal keys in the CHORAL system.

- ii, $ii_7 \rightarrow$ I; passing chord ii $\rightarrow$ I
- ii, $ii_7 \rightarrow$ vi, vi_7 in root position
- iii $\rightarrow$ IV, vi, vi_7
- IV, $IV_7 \rightarrow$ ii, ii_7, V, V_7, vii^o
- IV $\rightarrow$ I, IV_7
- passing chord IV $\rightarrow$ V
- V, $V_7 \rightarrow$ vi, ii, IV, iii, vii^o, I
- vi, $vi_7 \rightarrow$ IV, ii, V, V_7, iii
- $vii^o \rightarrow$ I
- ii, IV $\rightarrow I_c \rightarrow$ V, V_7
- The following chords: I, ii, ii_7, IV, V, V_7, vi, vi_7 may repeat

- The example from Bach's chorales: *O Welt, ich muss dich lassen*, No. T301 in Terry collection (No. R275, R363, R366 *O Welt, sieh hier dein Leben* in Riemenschneider collection) shows IV-IV_7 progression. It also illustrates that the continuity of the bass line seems to have more priority than the guideline on not to repeat the same harmony over the bar line.



- Summary of chord transition from left to right in a minor mode:

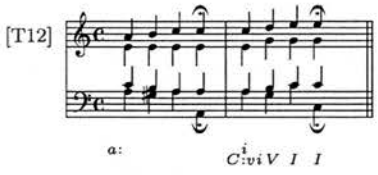
- i $\rightarrow$ ii^o, $ii^{\varnothing}_7$, ii, iv, iv_7
- i $\rightarrow$ IV, IV_7, V, V_7, v, VI
- i $\rightarrow$ vii^o, $vii^{\varnothing}_7$
- ii^o,$ii^{\varnothing}_7 \rightarrow$ V, V_7
- ii $\rightarrow$ i if some voice (not the bass voice) rises from the 5th of ii ($\flat\hat{6}$) to the root of i ($\hat{1}$)
- ii^o, $ii^{\varnothing}_7 \rightarrow$ i
- $III^+ \rightarrow$ i, VI
- iv, $iv_7 \rightarrow$ ii^o, $ii^{\varnothing}_7$, i, V, V_7
- iv, $iv_7 \rightarrow$ III^+, vii^o, $vii^{\varnothing}_7$
- IV, $IV_7 \rightarrow$ V, V_7
- V, $V_7 \rightarrow$ VI, vii^o, $vii^{\varnothing}_7$, i
- v $\rightarrow$ iv, iv_7, VI
- VI $\rightarrow$ iv, iv_7, ii^o, $ii^{\varnothing}_7$, V, V_7
- vii^o, $vii^{\varnothing}_7 \rightarrow$ i
- ii^o, $ii^{\varnothing}_7 \rightarrow I_c \rightarrow$ V, V_7

- iv, $iv_7 \rightarrow I_c \rightarrow V, V_7$
- $VI \rightarrow I_c \rightarrow V, V_7$
- The following chords: i, III, iv, iv_7, V, V_7, VI, VII may repeat

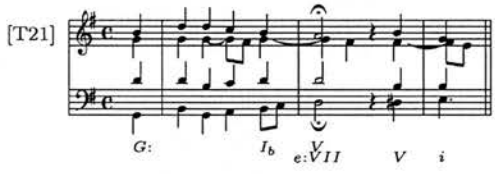
- If there is an accidental in the chord skeleton which does not agree with the current key in the CHORAL, the new key could be entered with the modulation patterns below, provided that: the accidentals of the chords preceding dominant or seventh chords must agree with the new key unless it is a subdominant sharp (e.g. f♯ in the key of C major or C minor); if the new key is a minor key, the movement of the melody line should agree with the melodic minor scale.

	Chords in new key	
Entry Type	Pivot chords	Entry chords
Tonic–Dominant	I	V, V_7
	i	V, V_7
Supertonic–Dominant	ii^o, ii^o_7	V,V_7
	ii, ii_7	V, V_7
Mediant–Dominant	III	V,V_7
Subdominant–Dominant	iv, iv_7	V, V_7
	IV	V, V_7
	$\sharp iv^o$, $\sharp iv^o_7$	V, V_7
	$\sharp iv^o_7$	V,V_7
Dominant–Dominant	V	V, V_7
Submediant–Dominant	VI	V, V_7
Leading–Dominant	vi, vi_7	V, V_7
	VII	V,V_7
Tonic–Leading	i	vii^o, vii^o_7
	I	vii^o
Mediant–Leading	III	vii^o
Subdominant–Leading	iv, iv_7	vii^o, vii^o_7
	IV	vii^o
Submediant–Leading	VI	vii^o, vii^o_7
	vi, vi_7	vii^o
Leading–Leading	♭VII	vii^o

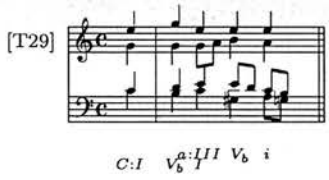
- An example of Bach's chorales: *Ach wie nichtig, ach wie flüchtig*—(T12, R48) shows a submediant–dominant (vi–V) modulation entry.



This example shows a descending minor third progression of roots (VII–V) modulation entry to a minor key. The example is from Bach's chorales: *Als der gütige Gott*—(T21, R159).



This example from Bach's chorales: *Auf meimen lieben Gott*—(T29, R304)[2]; the example shows a ascending major third progression of roots (III–V) modulation entry to a minor key.



- Ebcioglu also gives other modulation patterns employed by the CHORAL system:

Conditions	: Roots move with an ascending chromatic motion and with
	: the chord pattern of minor–diminish seventh–tonic chord
Pattern	: New key may be entered by e.g. Fm—F♯°—G (tonic)

- The modulation pattern seen in the new key:

[2]Note: There are discrepancies in harmonisations between Terry collection and Riemenschneider collection in these chorales.

Conditions : Next beat is a phrase ending
: Or alien accidental is ♭7 in the previous major key
: Or alien accidental is ♭2 in the previous minor key
: And the current key is not P4 above previous major key

Pivot chord	Entry chord	Remarks
I, V, V_7, vi, vi_7	ii, ii_7, IV, vi, vi_7	new key is a major key
i, VII, III	ii^o, $ii^{\varnothing}_7$ iv, iv_7, VI	new key is a minor key

- The modulation pattern seen in the new key:

Conditions : Alien accidental is ♭7 in the previous major key
: Or alien accidental is ♭2 in the previous minor key

Pivot chord	Entry chord	Remarks
iii, vii^o	V, V_7	new key is a major key

- The modulation pattern seen in the new key:

Conditions : Alien accidental is ♯4 in the previous major key
: Or alien accidental is ♯6 in the previous minor key

Pivot chord	Entry chord	Remarks
i, III, iv, iv_7, VI	ii^o, $ii^{\varnothing}_7$	new key is a minor key

- The modulation pattern seen in the new key:

Conditions : Alien accidental is ♯4 in the previous major key
: And it make the chord read vii instead of vii^o

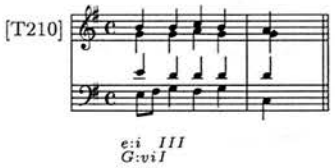
Pivot chord	Entry chord	Remarks
i, III, IV, IV_7, VII	ii	new key is a minor key

- The modulation pattern seen in the new key:

Conditions : New key is a relative major
: Both chord are in root positions

Pivot chord	Entry chord	Remarks
vi	I	

- The example from Bach's chorales: *Jesu meine Freude*—(T210, R263) shows a modulation from a minor key to its relative major key.



- Just before the phrase ending, a new key may be entered at the tonic:

  Conditions : Next beat is a phrase ending

  : And the current key is not P4 above previous major key

Pivot chord	Entry chord	Remarks
I, ii, IV, V, V_7	I	new key is a major key
i, ii^o, ii^o_7, III, iv, iv_7, VI, V, V_7, VII	i	new key is a minor key

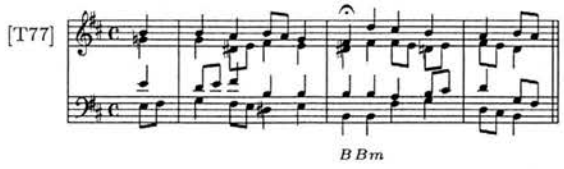
- Plagal entry to a new key in the sharps direction:

Previous chord	Entry chord	Remarks
ii	I	new key is a major key (see a)
ii	i	new key is a minor key (see a)
iv	i	new key is a minor key (see a)
IV	I	new key is a major key (see b)

a) Previous chord must be vi of a previous major key or i of a previous minor key.
This implies the modulation from:
tonic → dominant, subdominant → tonic, relative minor → dominant,
relative minor → relative dominant minor

b) Previous chord must be I of a previous major key or VI of a previous minor key.
This implies the modulation from:
tonic → dominant, subdominant → tonic, relative minor of dominant → tonic,
relative minor → subdominant

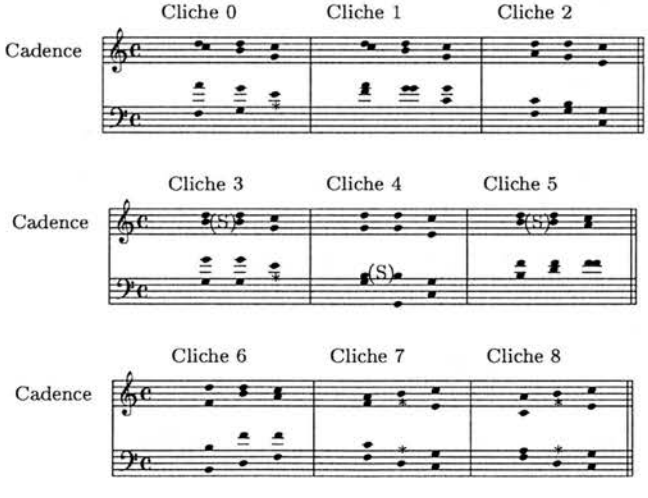
- If the last phrase ends with a major chord and a new phrase can start with the same chord in minor quality then a new phrase may be entered with a minor key with that minor chord as a tonic chord. The example below is from Bach's chorales: *Ein' feste Burg ist unser Gott*—(T77, R20). It shows a modulation to a new minor key in a new phrase.



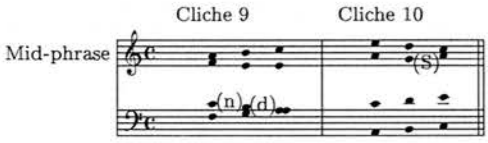
C.2.2.4 Generation of cliche (1.1.5)

There are two types of cliche; cadence cliches and mid phrase cliches. A cliche is selected if it matches three soprano skeleton notes. We present the cliches based on Ebcioglu's conventions:

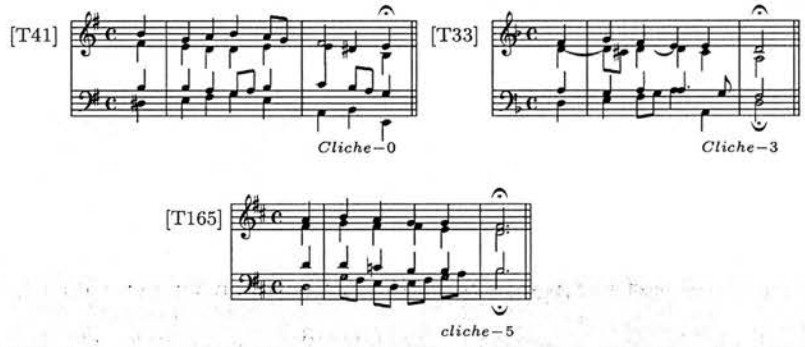
- Last soprano note is fixed on C5
- The symbol (*) means that the pitch is opened to more than one appropriate choice.
- The symbol (s) means that the pitch must be elaborated with a suspension at its strong quaver beat in the fill-in process (The fill-in process will be discussed later).
- The symbol (n) means that the pitch must be in a normal state in the fill-in process.
- The symbol (d) means that the pitch must be in a descending state in the fill-in process.
- Cadence cliches: There are nine cadence cliches in the CHORAL system (No 0 to No 8).



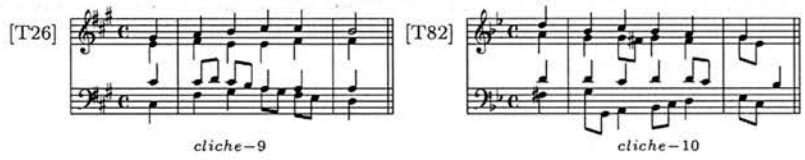
- Mid-phrase cliches: There are two mid-phrase cliches in the CHORAL system (No 9 and No 10).



- The uses of cliches are illustrated in the following examples from Bach's chorales[3]:
  - Cadence cliches: *Christ lag in Todesbanden*—(T41, R261); *Befiehl du deine Wege*—(T33, R340); *Herzlich tut mich verlangen*—(T165, R21)



  - Mid-phrase cliches: *Auf meinen lieben Gott*—(T26, R304); *Erhalt' uns, Herr, bei deinem Wort*—(T82, R72).



These examples suggest that the cliches are used with a tonal similarity context. They do not require the exact matches (e.g. A4–B4–C5 will match A4–B4–C♯5).

C.2.3 Constraint section in the Chord skeleton view (1.1.6)

C.2.3.1 Cadence constraints

- **Major mode**: The following cadence patterns are used in the major mode. The dominant seventh (V_7), diminish seventh (ii^o_7; $\sharp iv^o_7$; vii^o_7) and altered forms chords

[3]Note: There are discrepancies in harmonisations between the Terry collection and the Riemenschneider collection in examples T26 and T33.

(e.g. picardy 3rd) are also possible alternatives when appropriate. When the inversion is specified (e.g. V_a), the chord must be in that specified position.

Ending degree	Plausible Cadence patterns	Key
$\hat{1}$	V-I, vii^o-I, V_a-vi	Tonic
$\hat{1}$	V-I	Subdominant
$\hat{2}$	I-V, ii-V, IV-V	Tonic
$\hat{2}$	V-I, vii^o-I	Dominant
$\hat{2}$	V-I	Relative-minor of Subdominant
$\hat{3}$	V-I, vii^o-I	Tonic
$\hat{3}$	i-V, ii-V, IV-V	Relative-minor of Subdominant
$\hat{4}$	V-I, vii^o-I, V_a-vi	Subdominant
$\hat{4}$	V-I	Relative-minor of Subdominant
$\hat{5}$	I-V, ii-V, IV-V, IV-I	Tonic
$\hat{5}$	V-I, vii^o-I	Dominant
$\hat{5}$	V-I, vii^o-I	Relative-minor of Dominant
$\hat{6}$	V-I, V_a-VI	Relative-minor
$\hat{6}$	V-I	Subdominant
$\hat{7}$	I-V, ii-V, IV-V	Tonic
$\hat{7}$	V-I, vii^o-I	Dominant
$\hat{7}$	i-V, ii-V, IV-V	Relative-minor

- **Minor mode**: The following cadence patterns are used in the minor mode. The dominant seventh (V_7), diminish seventh (ii^o_7; $\sharp iv^o_7$; vii^o_7) and altered forms chords (e.g. picardy 3rd) are also possible alternatives when appropriate. When the inversion is specified (e.g. V_a), the chord must be in that specified position.

Ending degree	Plausible Cadence patterns	Key
$\hat{1}$	V-I, vii^o-I, V_a-VI	Tonic
$\hat{1}$	V-i, vii^o-i	Tonic
$\hat{2}$	i-V, ii^o-V, iv-V	Tonic
$\hat{2}$	V-I	Relative-major of Dominant
$\flat\hat{3}$	V-I, vii^o-I, V-i, vii^o-i	Tonic
$\flat\hat{3}$	V-I, V_a-vi	Relative-major

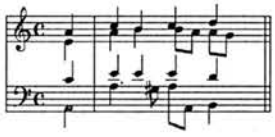
$\hat{4}$	V-I, V-i	Subdominant
$\hat{4}$	V-I, vii^o-I	Relative-major of Dominant
$\hat{5}$	i-V, ii^o-V, iv-V	Tonic
$\hat{5}$	V-I, vii^o-I, V-i, vii^o-i	Dominant
$\hat{5}$	i-V	Subdominant
$\hat{5}$	V-I	Relative-major
$\flat\hat{6}$	V-I, V_a-vi	Relative-major of Subdominant
$\flat\hat{7}$	V-I, vii^o-I	Relative-major of Dominant
$\flat\hat{7}$	V-I, vii^o-I, V-i, vii^o-i	Dominant
$\hat{7}$	i-V, ii^o-V, iv-V	Tonic

- In the deceptive cadence, the dominant must be in the root position. In the perfect cadence (with vii^o–i; vii^o–I) the leading chord must be in the first inversion. The plagal cadence is only allowed in the major key when the melody at the cadence falls from $\hat{6}$ to $\hat{5}$.
- The key is allowed to change at the final chord if it is seen as a perfect cadence in some keys.
- The v–V pattern is not allowed before the final chord.
- The III_b–V_a pattern is not allowed before the final chord.
- The VII–V pattern is not allowed before the final chord unless the soprano or the bass move by ascending chromatic $\flat\hat{7}$–$\hat{7}$.
- In both major and minor modes, the piece must end with perfect cadence with both chords in root positions. In a minor mode, the piece should end with a tonic major. The doubling of the final chord should be a root doubling.
- When the V chord appears in the first inversion before the final chord of a phrase. It should be filled in with an quaver note movement to reach the root.
- No voice may move more than a fifth leap in a cadence.
- Two consecutive phrases cannot both end with a deceptive cadence in the home key.
- The penultimate phrase cannot end with tonic in the home key.

C.2.3.2 Melodic constraints

- The melodic span of alto,tenor,bass cannot exceed octave + sixth

- A leap wider than sixth is not allowed but octave skip is allowed in bass
- A seventh skip is allowed in bass if preceded by a motion in the opposite direction (the motion has a skip less than sixth and the motion is not leading note-tonic).
- The leading bass-note of dominant chord or leading chord cannot jump octave.
- Leading note cannot progress up more than a fourth and down more than a third.
- Within a phrase, a melodic line cannot progress more than ninth (octave + second) in three notes; tenths in five notes.
- No augmented second, augmented fourth progression.
- V-VI progression in minor key (e.g. chord E – F in the key of A minor) the fifth of chord E which is $\hat{b}$ cannot progress to the root of chord F which is $\hat{f}$—a tritone progression.
- The bass note can jump over the bar line if :
  - It is at the beginning of anacrusis phrase
  - The second chord is seventh chord in third inversion X_{7d}

[T57] 

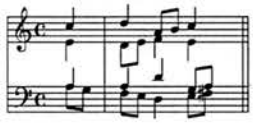
- The bass note cannot jump between the second and third beats if the two chords are the same.

C.2.3.3 Harmonic constraints

- The leading note in dominant and leading chord cannot be doubled (both major and minor key)
- The leading note in mediant chord cannot be doubled (minor key)
- During the modulation if the pivot chord can be seen as dominant or seventh chord in the previous key, then the leading note in that key cannot be doubled.
- In a progression from supertonic to mediant chord in a minor key, the root of supertonic cannot be doubled. (see next rule)
- The progression ii^o, ii^o_7 to III in minor key, if supertonic chord is in a root position then the mediant chord must be in root position (i.e. bass progress by step)
- The progression dominant,seventh to tonic in some key, if the leading note of the key is in the first chord then it must move to tonic of the second chord.
- The sharpen sixth in the melodic minor scale cannot be doubled.

- No consecutive fifth; octave progression
- No consecutive fifth; octave progression if separate by one intervening chord, when the intervening chord has the same root as one of its neighbours.
- No exposed fifth, octave (i.e. intervals of fifth and octave between the soprano and bass), except:
  - When the chord does not change
  - During the phrase ending when the soprano moves by step

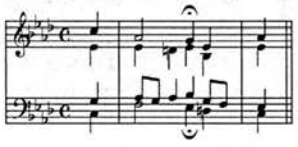
[T402]



Dm Dm

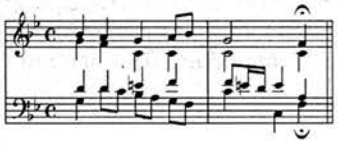
- Only V_7, vii°_7, ii°_7 are allowed to be used as seventh chords. Exceptions to the rules are when:
  - It is just two beats before the fermata, other seventh chords may be used.
  - Other Seventh chords are allowed at the very beginning of a phrase when the bass continues a linear progression.
  - A seventh chord in third inversion is allowed at the beginning of a bar if the bass repeats.

[T301]



V_{7d}

[T4]

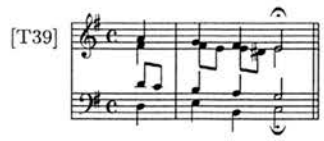


- Dissonant seventh pitch must be prepared and resolved.
- Dissonant seventh may not be prepared if :
  - When the chord does not change or
  - The dominant seventh chord is in the first inversion or
  - The seventh chord is a diminish seventh chord

C.2.4 Heuristic section in the Chord skeleton view (1.1.7)

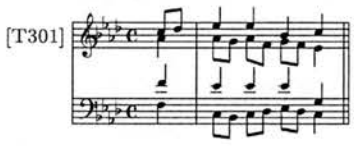
The list is in a decreasing order of priority

- Prefer to use deceptive cadence in the penultimate phrase, although the melody would fit a perfect cadence. This is to avoid consecutive perfect cadences.



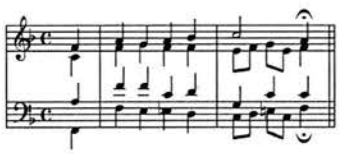
e:

- Prefer to use predefined cliches
- Prefer to continue the bass in one direction using
  - Ascending chromatic motion in the bass.
- Prefer to move by step in the bass.
- Prefer to move by third in the bass.
- Prefer to use the descending fourth cliche (i.e. bass descending fourth and make a 3–8 or 5–10 with some other voices that rise a third).



- If a pitch moves semitone in one voice and if the same pitch is also sounded in other voices, both should make a contrary motion.
- Avoid repetition by
  - Prefering to use available untried different bass notes. (window size = 10)
  - Prefering not to end the phrase with chord having the same root as the chord used in the previous phrase.
  - Prefering not to use the same phrase ending pitch used before.
  - Prefering to repeat a pitch in a high corner context.
- Prefer to have a skip (greater than 3rd) within a step movement in the opposite direction before and after a skip.
- Prefer to continue a linear progression in the tenor.
- Prefer to continue a linear progression in the alto.
- Prefer to move by step or third in the tenor.
- Prefer to move by step or third in the alto.
- Prefer to use different available tenor pitches.
- Prefer to use different available alto pitches.
- Prefer to have a skip (greater than 3rd) within a step movement in the opposite direction before and after a skip in tenor.

- Prefer to have a skip (greater than 3rd) within a step movement in the opposite direction before and after a skip in alto.
- Prefer not to have a parallel motion of all parts unless the target chord is a diminished seventh.
- Prefer to avoid chromatic motion in parts, except the ascending chromatic motion in the bass part.

[T48] 

- Prefer not to move skip-skip that makes a progression greater than a fifth.
- Prefer to have the first chord (thesis case) harmonised with tonic chord.
- Prefer not to use III^{+}
- Inversion type (this can be overridden by cliches)
  - Prefer to use root position triad (except diminished triad)
  - Prefer to use first inversion
- Chord type preferences
  - Prefer to use triads than sevenths
  - Prefer to use seventh chord for ii^{o}_{7},vii^{o}_{7} in minor key
- Prefer to be in home key in the first and the last phrase.
- Prefer to assert tonic after a new key is entered with dominant.
- Prefer progress by
  - Relative V-I; VII-I progression (strong cadence progression)
  - Relative II-I; IV-I progression
- Prefer modulation patterns
  - III-V, VII-V to a new minor key
  - vi-V to a new major key
- Prefer to avoid harmonic syncopation
- Prefer not to have unisons
- Prefer not to have exposed octaves and fifths
- Doubling preferences
  - Doubling fifth in second inversion
  - Doubling third in diminished chord
  - Doubling root in other chord types

- Doubling major third when a minor chord is in the root position and moves a step up.
- Doubling fifth

- Prefer not to have tritone progression in any three notes span.

C.2.5 Basic building blocks in the Fill-in process (2.1.1)

The fill-in process is a combination of the fill-in view, the melodic string view, the merged melodic string view and the time slice view.

C.2.6 Generation section in the Fill-in process (2.1.2)

The CHORAL program accepts melody input from users. The melody input is in the form of pitch strings (e.g. (la4 si4) do5 si4 do5 re5 ! mi 5 * *). Metrical structure and other information (e.g. anacrusis flag, if the pitch is taken to be a harmony note) are to be specified along with the melody input. The generation of the soprano attributes is from this input. The fill-in view observes the chorale as four interacting state machines that jump from state to state in lockstep [Ebcioglu, 1987, p268].

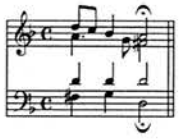
During the fill-in step n, the source chord is the skeleton chord of crotchet beat n and the target chord is the skeleton chord of crotchet beat $n+1$. The pitch of each voice is taken from the pitch from the chord skeleton. However, new pitches may be generated by the fill-in view. In this case the following accidentals are possible (i.e. c♯, d♯, f♯, g♯, and b♭—when the current key is C major or A minor).

C.2.6.1 Previous state is a normal state

First we will look at the generation for each of alto, tenor and bass voices at a normal state (or at a fresh start):

- General rule
  - Fill the crotchet beat n with the pitch from the chord skeleton n and
  - Fill the crotchet beat $n+1$ with the pitch from the chord skeleton $n+1$
- When the crotchet beats n and $n+1$ is a third apart (either ascending or descending)
  - Fill the passing quaver note between them

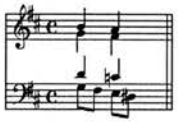
- When the crotchet beats n and $n+1$ form a descending second (see the alto voice in T7).
  - Fill the crotchet beat n with the pitch from the chord skeleton n and
  - Change the state to a suspension state

[T7] 

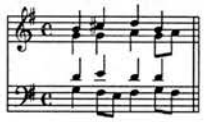
- When the crotchet beats n and $n+1$ form a descending fourth in the bass (see T165)

  -Fill the passing quaver note between them (which still need one more note) and

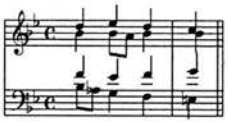
  -Change the state to a descending passing note state

[T165] 

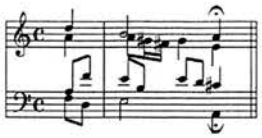
- When the crotchet beats n and $n+1$ form a descending third in the bass (see T210)
  - Fill the crotchet beat n with the pitch from the chord skeleton n and
  - Change the state to a descending passing note state

[T210] 

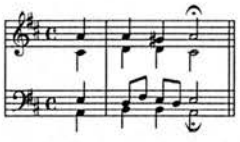
- When the crotchet beats n and $n+1$ have the same pitch
  - Fill the neighboring quaver note between them (see T174) or
  - Fill the upper neighboring quaver note and change to a suspension state

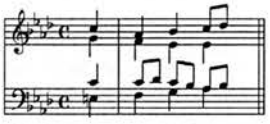
[174] 

- When the crotchet beats n and $n+1$ have the same pitch
  - Fill the leap fourth down quaver note between them (must be a consonance) and this must occur in accompaniment to a suspension (see T151).

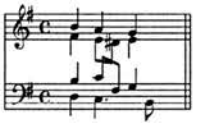
[T151] 

- When the crotchet beats n and $n+1$ have the same pitch and the crotchet beat n is a strong beat, the following may be tried (see T13).
  - Fill the crotchet beat n with two quaver notes (source and a third above)
  - Change the state to a descending passing note state
- The same idea could also be applied when the crotchet beats n and $n+1$ form a descending second and the crotchet beat n is a strong beat (see T301)
  - Fill the crotchet beat n with two quaver notes (source and a second above)
  - Change the state to a descending passing note state

[T13] 

[T301] 

- When the crotchet beats n and $n+1$ are more than a second apart but not a seventh, the crotchet beat n may be filled with two quaver notes: source and a step above the crotchet beat $n+1$ pitch (in case of an ascending progression); or source and a step below the crotchet beat $n+1$ pitch (in case of a descending progression).

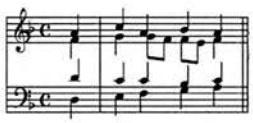
[T210] 

- When the crotchet beats n and $n+1$ form a second apart in the alto or tenor, the crotchet beat n may be filled with two quaver notes: source and a third above, or source and a fourth above. Provided that this does not violate other voice leading rules.
- When the crotchet beats n and $n+1$ form a second apart in the bass and the crotchet beat n is in a root position. The crotchet beat n may be filled with two quaver notes: source and a third above.

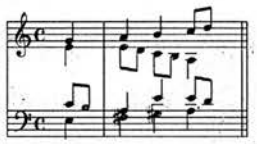
C.2.6.2 The previous state is a suspension state

Here we look at the generation for each voice (i.e. alto, tenor and bass) at a suspension state:

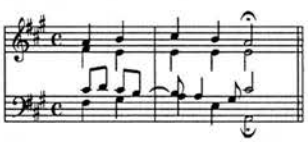
- General rule
  - Resolve the suspension on the weak quaver note of the crotchet beat n with the pitch from the chord skeleton n and
  - Change the state to a normal state.

[T22] 

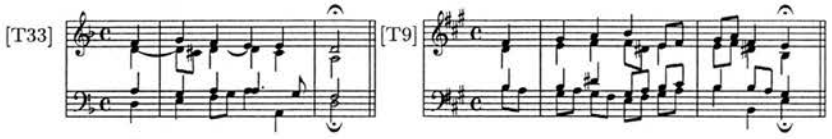
- When the crotchet beats n and $n+1$ form a descending third
  - Resolve the suspension on the weak quaver note of the crotchet beat n with the pitch from the chord skeleton n and
  - Change the state to a descending passing note state.

[T155] 

- When the crotchet beats n and $n+1$ form a descending second
  - Resolve the suspension on the weak quaver note of the crotchet beat n with the pitch from the chord skeleton n and
  - Continue the suspension state.

[T316] 

- When the crotchet beats n and $n+1$ have the same pitch
  - The suspension may be held for the crotchet beat n and to be resolved in the beat $n+1$.
  - Change the state to normal after resolving the suspension.
  - Another alternative may be to move a third down in the weak eight beat of the crotchet beat n and then move step up to the resolution pitch in the beat $n+1$.



C.2.6.3 The previous state is a descending passing note state

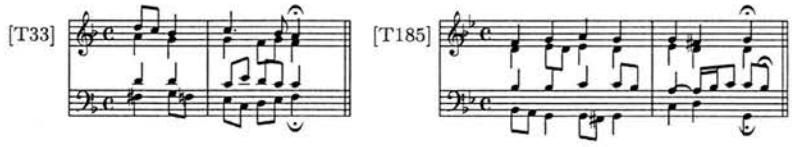
Here we look at the generation for each of alto, tenor and bass voices when the state is at a descending passing note state:

- Resolve the descending accented passing note on the weak quaver note of the crotchet beat n with the pitch from the chord skeleton n and change the state to a normal state.
- When the crotchet beats n and $n+1$ form a descending second
  - Resolve the dissonant on the weak quaver note of the crotchet beat n with the pitch from the chord skeleton n and
  - Continue the suspension state.
- When the crotchet beats n and $n+1$ form a descending third
  - Resolve the dissonant on the weak quaver note of the crotchet beat n with the pitch from the chord skeleton n and
  - Change the state to a descending passing note state.

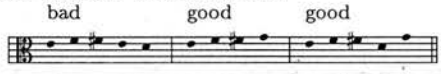
C.2.7 Constraint section in the Fill-in process (2.1.3)

General constraints in fill-in, melodic string, merged melodic string and time slice views are listed below:

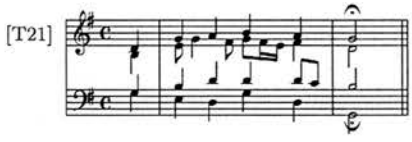
- Merged melodic string view: the pitch pattern x y x y is not allowed, unless it is enclosed in a sequence of w x y x y z in which both w x y and x y z form either both ascending or both descending patterns. However, Ebcioglu points out that the observed inner voices sometimes do not agree with this rule (see the alto voice in example T185).



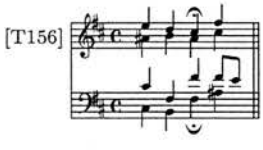
- Merged melodic string view: for the bass voice, a sequence of w x y x y z in which both w x y and x y z form either both ascending or both descending progression, the progression must be a scale progression.
- Merged melodic string view: do not repeat a high corner context in the same voice unless the high corners are more than eight crotchet beats apart.
- Melodic string view: do not repeat the same pitch more than three times.
- Melodic string view: do not leap larger than a sixth but octave leap is allowed.
- Melodic string view: Tritone spans over three notes are not allowed in the bass voice. Tritone spans over three notes in the alto or the tenor voice must be followed by a step in the same direction. Tritone spans over four notes in all voices (except soprano) must be preceded or followed by a step in the same direction.
- Merged melodic string view: In the bass, the chromatic motion is terminated with a scale degree in the same direction with at most one note intervening between the chromatic motion and its termination.

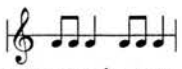


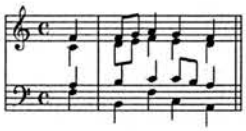
- Melodic string view: In the bass, three consecutive skips are allowed only if one of the three skips is a third skip; or a pair of non-consecutive pitches in the four pitches (three skips) has the same scale degree (e.g. C4 and C4 or C4 and C5).
- Melodic string view: Augmented and diminished intervals are not allowed, except diminished fourth and fifth which resolve down a step in the opposite direction.



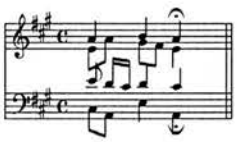
- Melodic string view: A sevenths, a diminished or augmented octave, or any interval greater than or equal to a ninth spanned in three notes in the same direction are not allowed. This does not apply to the bass line over the phrase boundary.



- Fill in view: Fill in the V_4–V_3 suspension in the perfect cadence progressed with V–V–I.
- Fill in view: The bass pattern  which starts on the strong beat cannot have two consecutive occurrences in any voices (except soprano). A pattern which appears from the combination of more than one voices is also not allowed (see T131). Two consecutive phrases are also not allowed to both begin with this pattern.

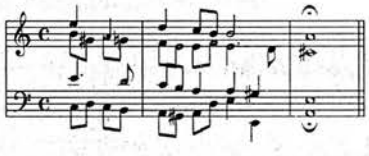
[T131] 

- Fill in view: In a jump up-down of three quaver notes between two crotchet beats (start on the strong quaver), the downward jump should be within the interval of a third.
- Fill in view: If the suspension is prepared via an inessential note at the weaker quaver, that quaver must form a chord. The inessential note must not appear in other voices (open octave).
- Fill in view: An interval of fourth formed by three quavers (e.g. D4-A3-D4) is used to accompany a suspension of a crotchet duration above it.
- Fill in view: If two voices simultaneously skip in the weaker quaver, it must constitute parallel thirds.
- Fill in view: No doubling of a leading note, doubling of a sharpen subdominant in a major key or doubling of a flatten submediant in a minor key are allowed.
- Fill in view: Never decorate the leading note in cadences. The example from Bach below is not allowed here (it is very rare).

[T17] 

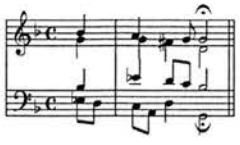
- Fill in view: Dissonant seventh should be prepared. The seventh produced from inessential notes in the weaker quaver beat must be treated properly in the next beat (i.e. resolved or suspended before resolving). The resolving seventh should not form 7-8 with the bass of each chord.

- Fill in view: In a perfect cadence progression, if the dominant chord is in the first inversion, it should be filled with the root in the weaker quaver. An exception to this rule is permitted when the root is approached by a descending fourth or a larger descending interval, or is approached by an ascending chromatic interval, and the phrase is not the last phrase of the choral.
- Fill in view: If the new key is entered at the cadence point, the new cadence should be a V_7-I progression. A seventh is needed to make the modulation sound convincing.
- Fill in view: In a supertonic–tonic progression, if the submediant pitch is moving to the tonic then it should be filled with a leading passing note.
- Fill in view: Begin each phrase with a normal state and end in a normal state. Each phrase should be self contained.
- Fill in view: The chromatic accidentals should be used in the right context. Here, we do not allow chromatic a neighboring note; the chromatic notes must agree with the tonality.
- Fill in view: If three or more voices are filled in the quaver weaker beat. It should be a meaningful chord, or the filled-in must be a passing note of each voice.

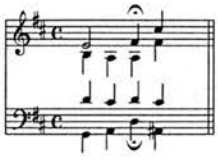
[T140] 

- Fill in view: The resolution of the suspension cannot occur in other voices above the voice effecting the suspension.
- Fill in view: If the chord is changed at the resolution of the suspended pitch then it must be a third below the root of the source chord.
- Fill in view: In the bass, a crotchet long suspension is not allowed.
- Fill in view: The suspension should be resolved on a weaker beat.
- Fill in view: If the suspension is prepared from an inessential note of the previous chord in a weak quaver then it should not be held longer than a crotchet (i.e. to be resolved in the next weak quaver beat).
- Time slice view: Consecutive fifth is allowed when the progression is from a perfect fifth to a diminished fifth and the parts move by ascending step.
- Time slice view: Consecutive fifth is allowed when the progression is from a diminished fifth to a perfect fifth formed by soprano and alto (or tenor) at the last chord of the phrase and the parts move by descending step.

- Time slice view: Consecutive fifth is allowed when the second perfect fifth appears before the final chord as a result of anticipation of the soprano pitch.

[T383] 

- Time slice view: False relations are not allowed

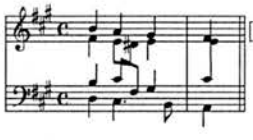
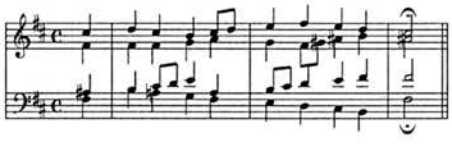
[T166] 

- Time slice view: Crossovers between voices are not allowed.

C.2.8 Heuristics section in the Fill-in process (2.1.4)

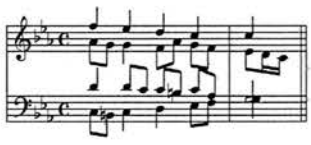
The list is presented in a decreasing order of priority.

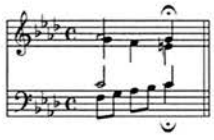
- Fill-in view: Prefer to use suspension in the bass only when it helps to hide the second inversion chord. In this case, the second inversion is on the weaker beat and the suspension is resolved by a semitone step.
- Melodic string view: In the bass, prefer to continue a linear progression in the bass and not to repeat the same pitch.

[T210]  [T99] 

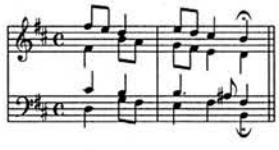
- Melodic string view: Prefer not to repeat the high corner pitch in the bass (i.e. prefer to maintain one high point)
- Fill-in view: Prefer to avoid the bass pattern  starting on the strong beat.
- Fill-in view: Avoid unison of inessential ornamentations
- Fill-in view: Avoid suspensions which do not create dissonant sonority (e.g. hold the pitch which will resolve to the fifth of the next chord).
- Fill-in view: Prefer a dissonant second suspension

- Fill-in view: Prefer a dissonant 9–8, 7–6, 4–3 suspension with the bass voice
- Fill-in view: Prefer repetitive suspension when applicable

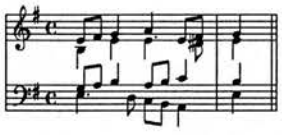
[T54] 

[T301] 

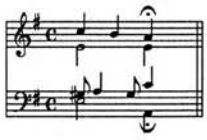
- Fill-in view: Avoid decorating the $\hat{5}$ scale degree between the V-I cadence
- Melodic string view: Prefer to continue a linear progression in the tenor
- Melodic string view: Prefer to continue a linear progression in the alto
- Merged melodic string view: Avoid repeating the high corners in the tenor
- Merged melodic string view: Avoid repeating the high corners in the alto
- Melodic string view: Prefer to move by step rather than by skip in the tenor
- Melodic string view: Prefer to move by step rather than by skip in the alto
- Time slice view: Prefer not to approach intervals of second, fourth and seventh with parallel motion.
- Time slice view: Prefer not to use exposed fifth, octave.
- Melodic string view: Prefer not to jump in the same direction of the scale movement (at least four pitches scale movement).
- Merged melodic string view: Avoid the pitch pattern x y x z x y x.
- Fill in view: It is undesirable to have an accented passing note with a dissonant second.
- Melodic string view: It is desirable to have a step before or after two consecutive skips.
- Melodic string view: Avoid tritones spanned in 3 or 4 notes.
- Fill in view: If the previous two beats do not have any quaver notes then it is desirable to have some.
- Time slice view: Parallel sixths and thirds are desired step wise movement between voices.

[T43] 

- Fill in view: The chord formed by the quaver notes fill in to form a smooth progression (e.g. V-I, VII-I).

[T39] 

- Merged melodic string view: Prefer to follow a leading note by the tonic note.

[T8] 

- Fill in view: After the resolution of a suspension, there should not be an upward skip.

Appendix D

Connections between Levels

D.1 Connections between Levels

Connections between the object-level and the meta-level are linked by a context mapping of ground terms between the two levels. The following prolog clause examples aim to illustrate how the contexts between the two levels are linked together. Each prolog clause reads

```
atomic control definition :- body of the atomic control definition
```

The body of the atomic control definition can be a single goal or many goals. The linking to the object level is via the body of the clause (Explanation of these atomic control definitions are included in chapter 6).

```
%%
rule( analyse(inputMelody),Ctrl,Ctrl,Score0,ScoreN ) :-
      group_soprano_line2phrases_with_harmonic_rhythm( Score0,ScoreN ).
%%
rule( selectPhrase(outlineHarmonicPlan),Ctrl,Ctrl,Score0,ScoreN ) :-
      select_phrase_hol( Score0,ScoreN ).
%%
rule( selectPhrase(outlineHarmonicPlan,Phrase),Ctrl,Ctrl,Score0,ScoreN ) :-
      Phrase = phrase-N, ground(N),
      select_phrase_hol( Phrase,Score0,ScoreN ).

%%
rule( outlineChord(Cadence),Ctrl,Ctrl,Score0,ScoreN ) :-
```

```
      ((Cadence == cadence -> outlineCadenceChord( Score0,ScoreN ));
       (Cadence == cadenceA -> outlineCadenceChorda( Score0,ScoreN ) )).
%%
rule( outlineChord(PropertyList),Ctrl,Ctrl,Score0,ScoreN ) :-
      is_list(PropertyList),
      (memberchk(cadence(CType),PropertyList) -> ground(CType);
       \+ memberchk(cadence(_),PropertyList)),
      (memberchk(key(KType),PropertyList) -> ground(KType);
       \+ memberchk(key(_),PropertyList)),
      outlineCadence( cadence(CType,KType),Score0,ScoreN ).
%%
test( property(state(bass),normal),Ctrl,Ctrl,Score ) :-
      bassNormalState( Score ).
%%
test( property(state(bass),suspension),Ctrl,Ctrl,Score ) :-
      bassSuspensionState( Score ).
%%
test( property(state(bass),descPassing),Ctrl,Ctrl,Score ) :-
      bassDescPassingState( Score ).
%%
measure( property(preferredChainedSuspension(alto)),Ctrl,Ctrl,Score0,Penalty ) :-
      preferChainedSuspensionAlto( Score0,Penalty ).
%%
measure( property(linearProgression(alto)),Ctrl,Ctrl,Score0,Penalty ) :-
      preferLinearProgressionAlto( Score0,Penalty ).
%%
measure( property(stepProgression(tenor)),Ctrl,Ctrl,Score0,Penalty ) :-
      preferStepTenor( Score0,Penalty ).
```

Bibliography

Ames, C. and Domino, M. (1992). Cybernetic composer: An overview. In M. Balaban, K. Ebcioglu, and O. Laske, editors, *Understanding Music with AI: Perspectives on music cognition*, chapter 8, pages 186–205. The AAAI Press/The MIT Press.

Athanasiou, E.-O. (1995). *Applying a Meta-level Architecture to a KBS for Harmonisation*. Master's thesis, Dept. of Artificial Intelligence, University of Edinburgh.

Baker, D. and Welsby, J. (1993). *Hymns and Hymn singing*. The Canterbury Press.

Balaban, M. (1992). Musical structures: Interleaving the temporal and hierarchical aspects in music. In M. Balaban, K. Ebcioglu, and O. Laske, editors, *Understanding Music with AI: Perspectives on music cognition*, chapter 5, pages 110–138. The AAAI Press/The MIT Press.

Baroni, M. and Jacoboni, C. (1978). *Proposal for a Grammar of Melody: The Bach Chorales*. Les Presses de l'Université de Montréal.

Baroni, M., Brunetti, R., Callegari, L., and Jacobni, C. (1982). A grammar for melody: Relationship between melody and harmony. In M. Baroni and L. Callegari, editors, *Musical grammar and Computer analysis*, pages 201–218. Firenze.

Bent, I. (1987). *Analysis*. Basingstoke : The Macmillan Press, Ltd. with a glossary by William Drabkin.

Blevis, E. B., Jenkins, M. A., and Glasgow, J. I. (1992). Motivations, sources, and initial design ideas for CALM: A composition analysis/generation language for music. In M. Balaban, K. Ebcioglu, and O. Laske, editors, *Understanding Music with AI: Perspectives on music cognition*, chapter 6, pages 141–154. The AAAI Press/The MIT Press.

Bowen, K. A. and Kowalski, R. (1982). Amalgamating language and metalanguage in logic programming. In K. Clark and S. Tarnlund, editors, *Logic programming*, pages 153–172. Academic Press. Also Research Paper Doc 81/30, Department of Computing, Imperial Colledge of Science and Technology, June 1981.

Boyd, M. (1967). *Harmonizing Bach Chorales*. Barrie and Jenkins Ltd.

Brachman, R. and Levesque, H. (1985). Readings in knowledge representation. In R. Brachman and H. Levesque, editors, *Readings in Knowledge Representation*. Morgan Kaufmann Publishers, Inc.

Bratko, I. (1990). *Prolog Programming for Artificial Intelligence*. Wokingham : Addison-Wesley.

Brook, F. J., Hopkins, A. L. J., Neumann, P. G., and Wright, W. V. (1993). An experiment in musical composition. In S. M. Schwanauer and D. A. Levitt, editors, *Machine Models of Music*, chapter 2, pages 23–40. The MIT Press.

Brunner, H. (1980). Barform. In S. Sadie, editor, *The New Grove Dictionary of Music and Musicians*, volume 2, page 156. London : Macmillan Publishers Ltd.

Buchanan, B. and Shortliffe, E. (1984). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley.

Bundy, A. (1987). What kind of field is artificial intelligence? In D. Partridge and Y. Wilks, editors, *Proceedings of the Workshop on the Foundations of Artificial Intelligence*. New Mexico. Also DAI Research Paper 305, Department of Artificial Intelligence, University of Edinburgh.

Bundy, A. (1990). *Cataloque of Artificial Intelligence Techniques*. Springer-Verlag.

Bundy, A. and Sterling, L. (1988). Meta-level inference: Two applications. *Journal of Automated Reasoning*, **4**(1), 15–28.

Bundy, A. and Welham, R. (1981). Using meta-level descriptions for selective application of multiple rewrite rules in algebraic manipulation. *Artificial Intelligence Journal*, **16**(2), 189–212. Also DAI Research Paper No.121, Dept Artificial Intelligence, University of Edinburgh.

Bundy, A., Byrd, L., Luger, G., Mellish, C., and Palmer, M. (1979). Solving mechanics problems using meta-level inference. In *Proceedings of the Sixth International Joint Conferences on Artificial Intelligence (IJCAI' 79)*. Also available in DAI Research Paper No.112, Dept Artificial Intelligence, University of Edinburgh.

Butterworth, A. (1994). *Stylistic Harmony*. Oxford University Press.

Buxton, W., Reeves, W., Baecker, R., and Mezei, L. (1978). The use of hierachy and instance in a data structure for computer music. *Computer Music Journal*, **2**(4), 10–20.

Clocksin, W. and Mellish, C. (1981). *Programming in Prolog*. Springer-Verlag, Berlin, Heidelberg, New York.

Cope, D. (1991). *Computers and Musical Style*. Oxford : Oxford University Press.

Courtot, F. (1992). Logical representation and induction for computer assisted composition. In M. Balaban, K. Ebcioglu, and O. Laske, editors, *Understanding Music with AI: Perspectives on music cognition*, chapter 7, pages 157–181. The AAAI Press/The MIT Press.

Dahlhaus, C. (1980). Harmony. In S. Sadie, editor, *The New Grove Dictionary of Music and Musicians*, volume 10. London : Macmillan Publishers Ltd.

Davis, R. and Buchanan, B. G. (1985). Meta-level knowledge overview and applications. In R. Brachman and H. Levesque, editors, *Readings in Knowledge Representation*, chapter 22, pages 390–397. Morgan Kaufmann Publishers, Inc.

Desain, P. and Honing, H. (1992). *Music, Mind and Machine : studies in computer music, music cognition and artificial intelligence*. Thesis Publishers Amsterdam.

Dunsby, J. and Whittall, A. (1988). *Music Analysis in Theory and Practice*. Faber Music Ltd.

Ebcioglu, K. (1987). Report on the choral project: An expert system for harmonizing four-part chorales. Technical report, IBM, Thomas J. Watson Research Center.

Ebcioglu, K. (1992). An expert system for harmonizing four-part chorales. In M. Balaban, K. Ebcioglu, and O. Laske, editors, *Understanding Music with AI: Perspectives on music cognition*, chapter 12, pages 294–333. The AAAI Press/The MIT Press.

Ebcioglu, K. (1993). An expert system for harmonizing four-part chorales. In S. M. Schwanauer and D. A. Levitt, editors, *Machine Models of Music*, chapter 17, pages 385–402. The MIT Press.

Edwards, P. (1967). *The Encyclopedia of philosophy*. New York, Macmillan.

Freeman, P. (1985). Knowledge elicitation–a commercial perspective. In *Proceedings of the First International Expert System Conference*. London, University of Reading, Reading, U.K.

Fry, C. (1991). Flavors band: A language for specifying musical style. In S. T. Pope, editor, *The Well-Tempered Object*, pages 49–63. The MIT Press.

Gammack, J. and Young, R. (1984). Psychological techniques for eliciting expert knowledge. In M. Bramer, editor, *R&D in Expert System, Proceedings of the 4th Expert System Conference*. CUP, London.

Gauldin, R. (1997). *Harmonic Practice in Tonal Music*. W.W. Norton and Company, Inc.

Harris, M., Wiggins, G., and Smaill, A. (1991). Representing music symbolically. In A. Camurri and C. Canepa, editors, *Proceedings of the IX Colloquio di Informatica Musicale*. Also Research Paper 562, Department of Artificial Intelligence, University of Edinburgh.

Hild, H., Feulner, J., and Menzel, W. (1991). Harmonet: A neural net for harmonizing chorales in the style of J.S. Bach. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing 4*, pages 267–274. Morgan Kaufman.

Hiller, L. (1970). Music composed with computers: A historical survey. In H. B. Lincoln, editor, *The Computer and Music*, chapter 4, pages 42–96. Cornell University Press.

Hiller, L. and Isaacson, L. (1993). Musical composition with a high-speed digital computer. In S. M. Schwanauer and D. A. Levitt, editors, *Machine Models of Music*, chapter 1, pages 9–22. The MIT Press.

Hindemith, P. (1942). *The Craft of Musical Composition*. Schott Music Corporation.

Holland, J. H. (1975). *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence*. Ann Arbor : The University of Michigan Press.

Holland, S. (1989). *Artificial Intelligence, Education and Music: The use of Artificial Intelligence to encourage and facilitate music composition by novices*. Ph.D. thesis, The Open University, Milton Keynes.

Ireland, A. (1992). The use of planning critics in mechanizing inductive proofs. In *International Conference on Logic Programming and Automated Reasoning—LPAR 92, St. Petersburg*, pages 178–189. Published as Lecture Notes in Artificial Intelligence No. 624, Springer-Verlag.

Ireland, A. and Bundy, A. (1996). Extensions to a generalization critic for inductive proof. In *13th Conference on Automated Deduction (CADE-13), pages 47–61*. Also in Lecture Notes in Artificial Intelligence No. 1104, Springer-Verlag.

Jamnik, M. (1999). *Automating Diagrammatic Proofs of Arithmetic Arguments*. Ph.D. thesis, Dept. of Artificial Intelligence, University of Edinburgh.

Korf, R. E. (1992). Search. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence Vol II*, pages 1460–1467. Wiley-Interscience Publication.

Krumhansl, C. L. (1990). *Cognitive Foundations of Musical Pitch*. New York, Oxford : Oxford University Press.

Leaver, R. A. and Bond, A. (1980). Luther. In S. Sadie, editor, *The New Grove Dictionary of Music and Musicians*, volume 11, pages 365–371. London : Macmillan Publishers Ltd.

Lerdahl, F. and Jackendoff, R. (1983). *A Generative Theory of Tonal Music*. The MIT Press.

Levesque, H. and Brachman, R. (1985). A fundamental tradeoff in knowledge representation and reasoning. In R. Brachman and H. Levesque, editors, *Readings in*

Knowledge Representation, chapter 4, pages 41–70. Morgan Kaufmann Publishers, Inc.

Longuet-Higgins, H. and Steedman, M. (1971). On interpreting Bach. *Machine Intelligence*, **6**, 221–241.

Longuet-Higgins, H. C. (1962). Letter to a musical friend. *The Music Review*, **23**, 244–248.

Loy, G. (1989). Composing with computers. In M. V. Mathews and J. R. Pierce, editors, *Current Directions in Computer Music Research*, chapter 21, pages 322–323. The MIT Press.

Lugar, G. F. and Stubblefield, W. A. (1989). *Artificial Intelligence and the Design of Expert System*. The Benjamin/Cummings Publishing Company Inc.

Marshall, R. L. (1972). *The Composition Process of J.S. Bach: A Study of the Autograph Scores of the Vocal Works*, volume 1–2. Princeton University Press.

Marshall, R. L. (1980a). Chorale. In S. Sadie, editor, *The New Grove Dictionary of Music and Musicians*, volume 4, pages 312–321. London : Macmillan Publishers Ltd.

Marshall, R. L. (1980b). Chorale settings. In S. Sadie, editor, *The New Grove Dictionary of Music and Musicians*, volume 4, pages 323–338. London : Macmillan Publishers Ltd.

Meltzer, B. (1970). Prolegomena to theory of efficiency of proof procedures. In N.V.Findler and B. Meltzer, editors, *Artificial Intelligence and Heuristic Programming*, chapter 2, pages 15–36. Edinburgh University Press.

Meyer, L. B. (1956). *Emotion and Meaning in Music*. The University of Chicago Press.

Meyer, L. B. (1967). *Music, the Arts and Ideas: Patterns and Predictions in Twentieth-century Music*. The University of Chicago Press.

Michalewicz, Z. (1992). *Genetic algorithms + data structures = evolution programs*. Berlin London : Springer-Verlag.

Mühlenbein, H. (1992). How genetic algorithms really work: Mutation and hillclimbing. In *Proceedings of Parallel Problem Solving from Nature, PPSN-92*, pages 15–26. Free University of Brussels.

Narmour, E. (1990). *The Analysis and Cognition of Basic Melodic Structure*. The University of Chicago Press.

Narmour, E. (1992). *The Analysis and Cognition of Melodic Complexity*. The University of Chicago Press.

Nilsson, N. J. (1971). *Problem Solving Method in Artificial Intelligence*. London : McGraw-Hill.

Palisca, C. V. (1980). Theory. In S. Sadie, editor, *The New Grove Dictionary of Music and Musicians*, volume 25, pages 359–385. London : Macmillan Publishers Ltd.

Phon-Amnuaisuk, S. (1997). *Four voice harmonisation for hymn tunes*. Master's thesis, Dept. of Artificial Intelligence, University of Edinburgh.

Phon-Amnuaisuk, S. and Wiggins, G. (1999). The Four-Part Harmonisation Problem: A comparison between Genetic Algorithms and A Rule-based System. In G. Wiggins, editor, *Proceedings of The AISB'99 Symposium on Musical Creativity*, pages 28–34. AISB.

Phon-Amnuaisuk, S., Tuson, A., and Wiggins, G. (1999). Evolving Musical Harmonisation. In *Proceedings of the Fourth International Conference on Neural Networks and Genetic Algorithms (ICANNGA'99), Slovenia*. Springer-Verlag.

Piston, W. (1982). *Harmony*. Victor Gollancz Ltd. Revised and expanded by Mark Devoto.

Pope, S. T. (1992). The SmOKe music representation, description language, and interchange format. In *ICMC Proceedings 1992*, pages 106–109. The Computer Music Association.

Pope, S. T. (1993). Music composition and editing by computer. In G. Haus, editor, *Music Processing*, chapter 2, pages 25–72. Oxford University Press.

Rader, G. M. (1993). A method for composing simple traditional music by computer. In S. M. Schwanauer and D. A. Levitt, editors, *Machine Models of Music*, chapter 11, pages 243–260. The MIT Press.

Reeves, C. R. (1993). Using genetic algorithms with small populations. In S. Forrest, editor, *Proceeding to the fifth International Conference on Genetic Algorithms*, pages 92–99. Morgan Kaufmann Publishers.

Reti, R. (1951). *The Thematic Process in Music*. Green wood Press, Westport Connecticut.

Riemenschneider, A. (1941). *371 Harmonized Chorales and 69 Chorale Melodies with Figured Bass*. G. Schirmer, Inc.

Rothgeb, J. (1993). Simulating musical skills by digital computer. In S. M. Schwanauer and D. A. Levitt, editors, *Machine Models of Music*, chapter 7, pages 157–166. The MIT Press.

Rowe, N. C. (1988). *Artificial Intelligence Through Prolog*. Prentice-Hall International, Inc.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence, A Modern Approach*. Prentice Hall.

Sadie, S. (1980). *The New Grove Dictionary of Music and Musicians*. London : Macmillan Publishers Ltd.

Schoenberg, A. (1967). *Fundamentals of musical composition*. Faber and Faber Ltd. edited by Gerald Strang with the collaboration of Leonard Stein.

Schoenberg, A. (1990). *Theory of Harmony*. Faber and Faber Ltd. translated by Roy R. Carter.

Schottstaedt, W. (1989). Automatic counterpoint. In M. V. Mathews and J. R. Pierce, editors, *Current Directions in Computer Music Research*, chapter 16, pages 199–214. The MIT Press.

Selfridge-Field, E. (1997). *Beyound MIDI*. The MIT Press.

Sloboda, J. (1985). *The Musical Mind: The Cognitive Psychology of Music*. Oxford University Press.

Smaill, A., Wiggins, G., and Harris, M. (1993a). Hierarchical music representation for composition and analysis. *Computers and the Humanities*, **27**, 7–17. Also Research Paper 511, Department of Artificial Intelligence, University of Edinburgh.

Smaill, A., Wiggins, G., and Miranda, E. (1993b). Music representation: Between the musician and the computer. In M. Smith, G. Wiggins, and A. Smaill, editors, *Music education : an artificial intelligence approach; Proceedings of a Workshop held as part of AI-ED 93, World Conference on AI in Education, Edinburgh*. Springer-Verlag. Also Research Paper 668, Department of Artificial Intelligence, University of edinburgh.

Smith, B. C. (1985). Prologue to reflection and semantics in a procedural language. In R. Brachman and H. Levesque, editors, *Readings in Knowledge Representation*, chapter 3, pages 31–40. Morgan Kaufmann Publishers, Inc.

Smoliar, S. W. (1993). Process structuring and music theory. In S. M. Schwanauer and D. A. Levitt, editors, *Machine Models of Music*, chapter 9, pages 188–212. The MIT Press.

Stacy, C. M. and Charles, F. S. (1992). On the use of problem reduction search for automated music composition. In M. Balaban, K. Ebcioglu, and O. Laske, editors, *Understanding Music with AI: Perspectives on music cognition*, chapter 10, pages 238–256. The AAAI Press/The MIT Press.

Steedman, M. (1984). A generative grammar for jazz chord sequences. *Music Perception*, **2**, 52–77.

Stefik, M. (1995). *Introduction to Knowledge Systems*. San Francisco : Morgan Kaufmann Publishers, Inc.

Sterling, L. and Shapiro, E. (1994). *The Art of Prolog*. MIT Press, 4th edition.

Sterling, L., Bundy, A., Byrd, L., O'Keefe, R., and Silver, B. (1982). Solving symbolic equations with press. In J. Calmet, editor, *Computer Algebra*. Lecture Notes in Computer Science No. 144, Springer Verlag, 1982. Also DAI Research Paper 171, Department of Artificial Intelligence, University of Edinburgh.

Tanslay, D. and Hayball, C. (1993). *Knowledge-based Systems Analysis and Design*. Prentice Hall.

Taylor, E. (1989). *The AB guide to music theory*. The Associated Board of the Royal Schools of Music (publishing) Ltd.

Tovey, D. F. (1935). *A Companion to Beethoven's Pianoforte Sonatas*. The Associated Board of The Royal Schools of Music.

Turek, R. (1988). *The Elements of Music: Concepts and Applications*. Alfred A. Knopf, New York.

Tusler, R. L. (1968). *The Style of J.S. Bach's Chorale Preludes*. Da Capo Press.

van Harmelen, F. (1989a). A classification of meta-level architecture. In P. Jackson, H. Reichgelt, and F. van Harmelen, editors, *Logic-based knowledge representation*, chapter 2, pages 13–36. The MIT Press.

van Harmelen, F. (1989b). *On the Efficiency of Meta-Level Inference*. Ph.D. thesis, Dept. of Artificial Intelligence, University of Edinburgh.

Way, E. C. (1994). *Knowledge representation and metaphor*. Intellect Books.

West, R., Howell, P., and Cross, I. (1991). Musical structure and knowledge representation. In P. Howell, R. West, and I. Cross, editors, *Representing Musical Structure*, chapter 1, pages 1–30. Academic Press.

Weyhrauch, R. W. (1985). Prolegomena to a theory of formal reasoning. In R. Brachman and H. Levesque, editors, *Readings in Knowledge Representation*, chapter 16, pages 309–328. Morgan Kaufmann Publishers, Inc. Also Research Paper STAN-CS-78-687, Department of Computer Science, Stanford University.

Wiggins, G., Harris, M., and Smaill, A. (1989). Representing music for analysis and composition. In *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI'89)*. Also Research Paper 504, Department of Artificial Intelligence, University of Edinburgh.

Wiggins, G., Miranda, E., Smaill, A., and Harris, M. (1993). A framework for the evaluation of music representation systems. *Computer Music Journal*, **17**(3).

Winograd, T. (1985). Frame representation and the declarative–procedural controversy. In R. Brachman and H. Levesque, editors, *Readings in Knowledge Representation*, chapter 22, pages 357–370. Morgan Kaufmann Publishers, Inc.

Wittlich, G. E. and Martin, D. S. (1989). *Tonal Harmony for the Keyboard*. Schirmer Books.